UCI ML Repo - credit card defaults

Training

Platform: Python 3, colab.research.google.com

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from joblib import dump, load
from google.colab import drive
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, log_loss
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

Load data

```
drive.mount('/content/gdrive', force_remount=False)
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive"
```

```
loc = "/content/gdrive/My Drive/Colab Notebooks/uci-credit-card-defaults/data/defaults_clean.csv"
data = pd.read_csv(loc, header=0)
data.shape
```

```
(30000, 25)
```

```
data.head(5)
```

	id	limit_bal	sex	education	marriage	age	pay_0	pay_2	pay_3	pay_4	...	bill_amt4	bill_amt5	bill_ar
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19

5 rows × 25 columns

```
data.tail(5)
```

	id	limit_bal	sex	education	marriage	age	pay_0	pay_2	pay_3	pay_4	...	bill_amt4	bill_amt5
29995	29996	220000	1	3	1	39	0	0	0	0	...	88004	31237
29996	29997	150000	1	3	2	43	-1	-1	-1	-1	...	8979	5190
29997	29998	30000	1	2	2	37	4	3	2	-1	...	20878	20582
29998	29999	80000	1	3	1	41	1	-1	0	0	...	52774	11855
29999	30000	50000	1	2	1	46	0	0	0	0	...	36535	32428

5 rows × 25 columns

```
type_dict = {}
for i in ["limit_bal", "bill_amt1", "bill_amt2", "bill_amt3", "bill_amt4",
          "bill_amt5", "bill_amt6", "pay_amt1", "pay_amt2", "pay_amt3",
          "pay_amt4", "pay_amt5", "pay_amt6"]:
    if i in data.columns:
        type_dict[i] = "float64"
data = data.astype(type_dict)
data.dtypes
```

```
id             int64
limit_bal    float64
sex            int64
education      int64
marriage       int64
age            int64
pay_0          int64
pay_2          int64
pay_3          int64
pay_4          int64
pay_5          int64
pay_6          int64
bill_amt1    float64
bill_amt2    float64
bill_amt3    float64
bill_amt4    float64
bill_amt5    float64
bill_amt6    float64
pay_amt1     float64
pay_amt2     float64
pay_amt3     float64
pay_amt4     float64
pay_amt5     float64
pay_amt6     float64
default        int64
```

```
1 data.describe()
```

	id	limit_bal	sex	education	marriage	age	pay_0	pay_2
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	15000.500000	167484.322667	1.603733	1.842267	1.557267	35.485500	-0.016700	-0.133767

Training

min	1.000000	10000.000000	1.000000	1.000000	1.000000	21.000000	-2.000000	-2.000000

```
def view_metrics(y_test, y_pred):
    print("Accuracy: {}".format(accuracy_score(y_test, y_pred)))
    print("Precision: {}".format(precision_score(y_test, y_pred)))
    print("Recall: {}".format(recall_score(y_test, y_pred)))
    print("F1: {}".format(f1_score(y_test, y_pred)))
```

```
# split train/test
y = data.loc[:, "default"]
X = data.drop(["id", "default"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
loc_data = "/content/gdrive/My Drive/Colab Notebooks/uci-credit-card-defaults/data"
# save test file for evaluation
X_test.to_csv("{}/defaults_clean_test_X.csv".format(loc_data), index=False)
y_test.to_csv("{}/defaults_clean_test_y.csv".format(loc_data), index=False)
X_train.head(5)
```

	limit_bal	sex	education	marriage	age	pay_0	pay_2	pay_3	pay_4	pay_5	...	bill_amt3	bill_amt4
28465	240000.0	2	1	1	40	-2	-2	-2	-2	-2	...	0.0	0.0
27622	50000.0	2	1	2	23	-1	-1	-1	-1	-1	...	2299.0	4800.0
28376	50000.0	2	2	1	36	2	2	2	2	0	...	49125.0	47956.0
10917	200000.0	2	3	1	54	6	5	4	3	2	...	104686.0	102549.0
27234	240000.0	1	1	1	35	-1	-1	-1	0	-1	...	21790.0	17102.0

5 rows × 23 columns

```
transformed_features = None
```

Tranformation pipeline classes

```
lass StdCol():
    """
    Standardizes column 'col_name' in a pipeline
    """
    def __init__(self, col_name):
        self.col_name = col_name

    def fit(self, X, y=None):
        self.ss = StandardScaler()
        self.ss.fit(X.loc[:, self.col_name].values.reshape(-1, 1))
        return self

    def transform(self, X):
        X = X.copy(deep=True)
        X.loc[:, self.col_name] = self.ss.transform(X.loc[:, self.col_name].values.reshape(-1, 1))
        global transformed_features
        transformed_features = X
        return X
```

```
class LogCol():
    """
    Log transforms column 'col_name' in a pipeline
    """
    def __init__(self, col_name):
        self.col_name = col_name

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy(deep=True)
        X.loc[X[self.col_name]<1, self.col_name] = 1
        X.loc[:, self.col_name] = np.log(X.loc[:, self.col_name])
        global transformed_features
        transformed_features = X
        return X
```

```
class CategoricalColInt():
    """
    Tranforms column 'col_name' into n-1 categorical columns
    """
    def __init__(self, col_name):
```

```
        self.col_name = col_name

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy(deep=True)
        dummies = pd.get_dummies(X.loc[:, self.col_name], prefix=self.col_name)
        dummies_cols = list(dummies.columns)[1:] # drop last new category to avoid feature correlation
        X = X.merge(dummies.loc[:,dummies_cols], left_index=True, right_index=True)
        X = X.drop(columns=self.col_name, axis=1)
        global transformed_features
        transformed_features = X
        return X
```

```
class AveColumns():
    """
    Calc average of columns
    """
    def __init__(self, feature_name, cols):
        self.feature_name = feature_name
        self.cols = cols
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy(deep=True)
        X.loc[:, self.feature_name] = X.loc[:, self.cols].mean(axis=1)
        global transformed_features
        transformed_features = X
        return X
```

```
class StDevColumns():
    """
    Calc stardard deviation of columns cols
    """
    def __init__(self, feature_name, cols):
        self.feature_name = feature_name
        self.cols = cols
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X = X.copy(deep=True)
        X.loc[:, self.feature_name] = X.loc[:, self.cols].std(axis=1)
```

```
        global transformed_features
        transformed_features = X
        return X
```

```
class SelectKBestFeatures():
    """
    Selects num of features to K_best
    """
    def __init__(self):
        pass

    def fit(self, X, y=None):
        self.selector = SelectKBest(k=20).fit(X, y)
        return self

    def transform(self, X):
        X = X.copy(deep=True)
        return self.selector.transform(X)
```

Train pipeline

```
pipe_list = []
ave_cols = {"avg_bill": ["bill_amt1", "bill_amt2", "bill_amt3", "bill_amt4", "bill_amt5", "bill_amt6"],
            "avg_pay": ["pay_amt1", "pay_amt2", "pay_amt3", "pay_amt4", "pay_amt5", "pay_amt6"]}
st_dev_cols = {"st_dev_bill": ["bill_amt1", "bill_amt2", "bill_amt3", "bill_amt4", "bill_amt5", "bill_amt6"],
            "st_dev_pay": ["pay_amt1", "pay_amt2", "pay_amt3", "pay_amt4", "pay_amt5", "pay_amt6"]}
categorical_cols = ["sex", "education", "marriage"]
log_cols = ["limit_bal", "avg_bill", "avg_pay",
            "bill_amt1", "bill_amt2", "bill_amt3", "bill_amt4", "bill_amt5", "bill_amt6",
            "pay_amt1", "pay_amt2", "pay_amt3", "pay_amt4", "pay_amt5", "pay_amt6"]
standardize_cols = ["age"]
for i in ave_cols:
    pipe_list.append(("enc_ave_"+i, AveColumns(i, ave_cols[i])))
for i in st_dev_cols:
    pipe_list.append(("enc_stdev_"+i, StDevColumns(i, st_dev_cols[i])))
for i in categorical_cols:
    if i in X_train.columns:
        pipe_list.append(("enc_cat_"+i, CategoricalColInt(col_name=i)))
for i in log_cols:
    if i in X_train.columns:
        pipe_list.append(("enc_log_"+i, LogCol(col_name=i)))
for i in standardize_cols:
    if i in X_train.columns:
        pipe_list.append(("enc_std_"+i, StdCol(col_name=i)))
pipe_list.append(("k_best_selector", SelectKBestFeatures()))
pipe_list.append(("model", LogisticRegression()))
pipeline = Pipeline(pipe_list)
```

```
model = pipeline.fit(X_train, y_train)
transformed_features.columns
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with inp
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with inp
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver w
  FutureWarning)
Index(['limit_bal', 'age', 'pay_0', 'pay_2', 'pay_3', 'pay_4', 'pay_5',
       'pay_6', 'bill_amt1', 'bill_amt2', 'bill_amt3', 'bill_amt4',
       'bill_amt5', 'bill_amt6', 'pay_amt1', 'pay_amt2', 'pay_amt3',
       'pay_amt4', 'pay_amt5', 'pay_amt6', 'avg_bill', 'avg_pay',
       'st_dev_bill', 'st_dev_pay', 'sex_2', 'education_2', 'education_3',
       'education_4', 'marriage_2', 'marriage_3'],
      dtype='object')
```

```
y_pred = model.predict(X_train)
view_metrics(y_train, y_pred)
```

```
Accuracy: 0.8031428571428572
Precision: 0.6782894736842106
Recall: 0.22048759623609923
F1: 0.3327953518398967
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with inp
  warnings.warn(msg, DataConversionWarning)
```

```
# parameters = {
#     "model__penalty": ["l2"],
#     "model__solver": ["lbfgs", "liblinear"],
#     "model__max_iter": [50, 100],
#     "model__C": [0.7, 0.3, 0.1]}
# grid = GridSearchCV(pipeline, parameters, cv=4, scoring="f1")
# grid.fit(X_train, y_train)
# print(grid.best_params_)
# y_pred = grid.predict(X_train)
# view_metrics(y_train, y_pred)
```

```
# print("y_train")
# print(np.array(y_train)[:200])
# print("y_pred")
# print(y_pred[:200])
```

```
# loc_model = "/content/gdrive/My Drive/Colab Notebooks/uci-credit-card-defaults"
# dump(grid.best_estimator_, "{}/model.joblib".format(loc_model))
```

```
```