Credit Risk Prediction web app by Flask

LISUM10

By: Mohsen Bahremani

June 26, 2022

https://github.com/MohsenBah/Credit_risk_prediction

Agenda

- ✓**Goal**: Prediction of Credit Risk – a classification model
- ✓ **Data**: it has 9 features and 1 response variable: **Age, Sex, Job, Housing, Saving accounts, Checking accounts, Credit amount , Duration, Purpose, Risk** (target value)
- ✓ **Model:** some models have been trained to predict whether a customer is defaulter or not.
- ✓**Deployment:** Logistic Regression has been selected to deploy web app by Flask API.

Data

In [2]:
```
data=pd.read_csv("/Users/mohsen/Documents/GitHub/Credit_risk_prediction/german_credit_data.csv", index_col=0)
data
```

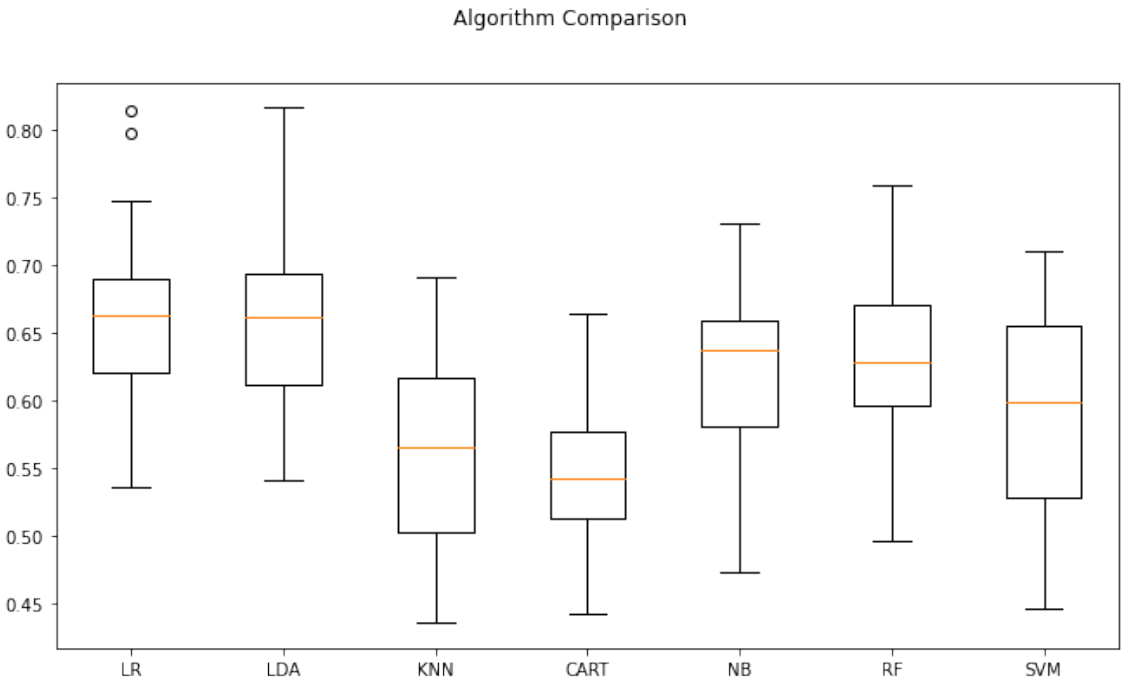
Out[2]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	67	male	2	own	NaN	little	1169	6	radio/TV	good
1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
2	49	male	1	own	little	NaN	2096	12	education	good
3	45	male	2	free	little	little	7882	42	furniture/equipment	good
4	53	male	2	free	little	little	4870	24	car	bad
...	...	...	...	...	...	...	...	...	...	...
995	31	female	1	own	little	NaN	1736	12	furniture/equipment	good
996	40	male	3	own	little	little	3857	30	car	good
997	38	male	2	own	little	NaN	804	12	radio/TV	good
998	23	male	2	free	little	little	1845	45	radio/TV	bad
999	27	male	2	own	moderate	moderate	4576	45	car	good

1000 rows × 10 columns

Model selection


```
seed = 123

# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC(gamma='auto')))

# evaluate each model in turn
results = []
names = []
scoring = 'roc_auc'
print("model, AUC mean, AUC STD")
for name, model in models:
        kfold=RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
        cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

# boxplot algorithm comparison
fig = plt.figure(figsize=(11,6))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Saving (serialization) the final Model


```
params = {
    "C": np.logspace(-3, 3, 7),
    "penalty": ["l1", "l2"]
}  # l1 lasso l2 ridge
model = LogisticRegression()
kfold=RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

logreg_cv = GridSearchCV(model, params, cv=kfold)
logreg_cv.fit(X, y)
print("tuned hpyerparameters :(best parameters) ", logreg_cv.best_params_)
print("accuracy :", logreg_cv.best_score_)
```

Logistic Regression has been selected as the final model. The grid search chose l2 penalty.

The model was saved as a **Pickle file.**

```
pickle.dump(logreg_cv, open('model.pkl','wb'))
```

Web App Script

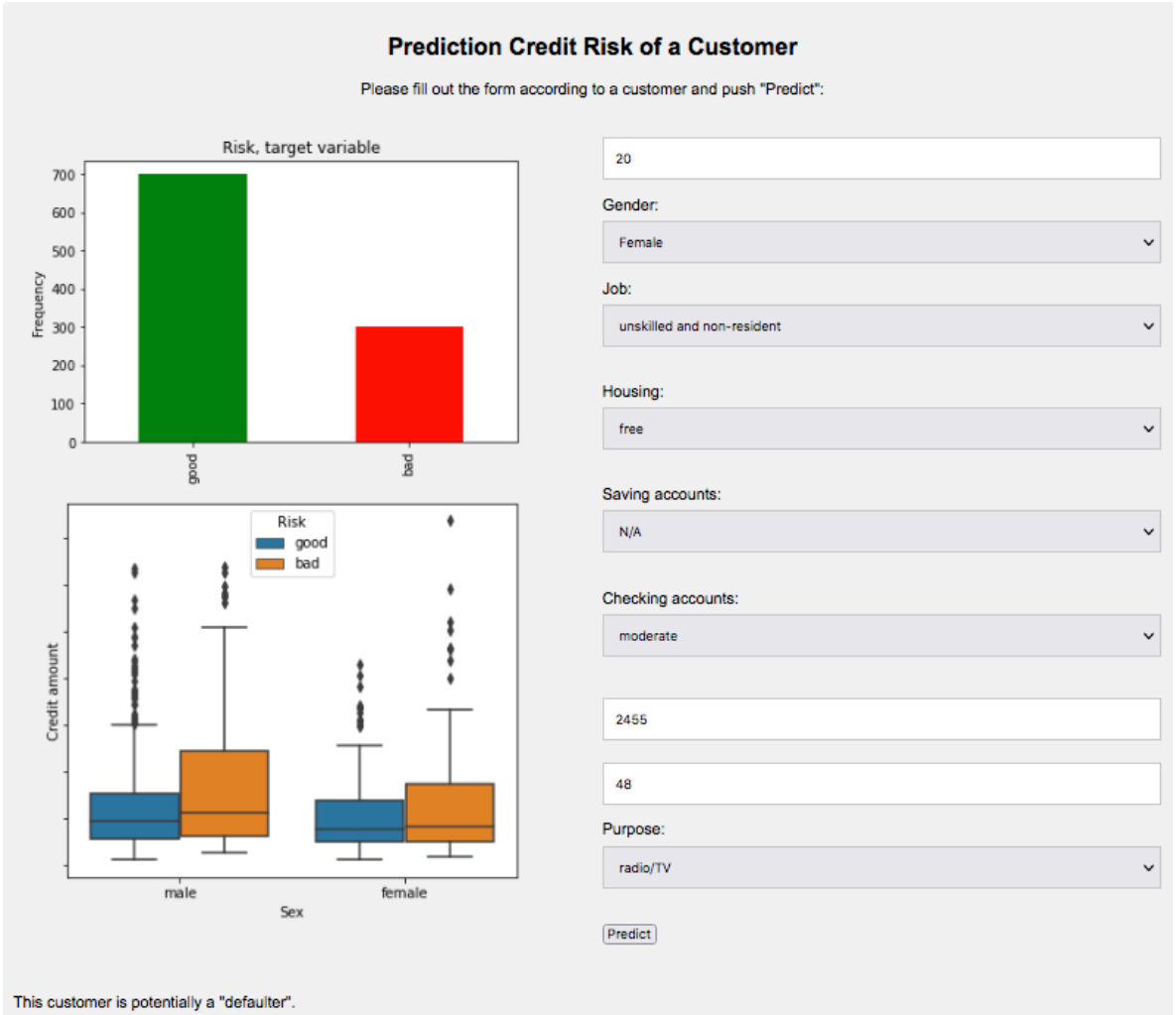
Web app via Flask API was written as shown.

The new test observation is needed to some preprocessing to become ready to predict, including changing the categorical variables to dummies.

```
import numpy as np
import pandas as pd
from flask import Flask, request, render_template
import pickle
import os


app = Flask(__name__)


clf = pickle.load(open('/Users/mohsen/Documents/GitHub/Credit_risk_prediction/model.pkl','rb'))


@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
    features = [x for x in request.form.values()]
    a = [np.array(features)]
    b=pd.DataFrame(a,columns=['Age', 'Sex', 'Job', 'Housing', 'Saving accounts',
                              'Checking account','Credit amount', 'Duration',
                              'Purpose'
                              ])
    c=pd.get_dummies(b,columns=[
                       'Sex', "Housing", 'Saving accounts', 'Checking account',
                       'Purpose'
                   ])
    new_x=pd.DataFrame(np.zeros((1,19)),columns=['Age', 'Job', 'Credit amount', 'Duration', 'Sex_male', 'Housing_own',
       'Housing_rent', 'Saving accounts_moderate',
       'Saving accounts_quite rich', 'Saving accounts_rich',
       'Checking account_moderate', 'Checking account_rich', 'Purpose_car',
       'Purpose_domestic appliances', 'Purpose_education',
       'Purpose_furniture/equipment', 'Purpose_radio/TV', 'Purpose_repairs',
       'Purpose_vacation/others'])
    for i in c.columns:
        if i in new_x.columns:
           new_x[i]=1
    pred = clf.predict(new_x)[0]
    if pred == 1:
       output = "non defaulter"
    else:
       output = "defaulter"


    return render_template('index.html', prediction_text='This customer is potentially a "{}".'.format(output))

if __name__ == "__main__":
    app.run(debug=True)
```

Web app screenshot with a sample customer

- Finally, the model was deployed.
- The numeric variables are needed to input manually, but categorical variables are drop down menu.
- This customer with the given information as shown is potentially a defaulter.



Thank You