# Week9assignment

October 25, 2024

1. Import the dataset and ensure that it loaded properly.
2. Prepare the data for modeling by performing the following steps:

a. Drop the column "Load_ID."
b. Drop any rows with missing data.
c. Convert the categorical features dummy variables.

3. Split the data into a training and test set, where the "Loan_Status" column is the target.
4. Create a pipeline with a min-max scaler and a KNN classifier (see section 15.3 in the Machine Learning with Python Cookbook).
5. Fit a default KNN classifier to the data with this pipeline. Report the model accuracy on the test set. Note: Fitting a pipeline model works just like fitting a regular model.
6. Create a search space for your KNN classifier where your "n_neighbors" parameter varies from 1 to 10. (see section 15.3 in the Machine Learning with Python Cookbook).
7. Fit a grid search with your pipeline, search space, and 5-fold cross-validation to find the best value for the "n_neighbors" parameter.
8. Find the accuracy of the grid search best model on the test set. Note: It is possible that this will not be an improvement over the default model, but likely it will be.
9. Now, repeat steps 6 and 7 with the same pipeline, but expand your search space to include logistic regression and random forest models with the hyperparameter values in section 12.3 of the Machine Learning with Python Cookbook.
10. What are the best model and hyperparameters found in the grid search? Find the accuracy of this model on the test set.
11. Summarize your results.

[2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

[ ]:
```

```

[ ]:
```

```

1. Import the dataset and ensure that it loaded properly.

[4]:
```
df = pd.read_csv('Loan_Train.csv')
```

[5]:
```
df.head()
```

[5]:
```
     Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

[6]:
```python
df.shape
```

[6]:
```
(614, 13)
```

2. Prepare the data for modeling by performing the following steps:

a. Drop the column “Load_ID.”
b. Drop any rows with missing data.
c. Convert the categorical features into dummy variables.

[ ]:
```python

```

[8]:
```python
df.drop('Loan_ID',axis=1,inplace=True)
```

[9]:
```python
df.isnull().sum()
```

[9]:
```
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
```

```
Loan_Status          0
dtype: int64
```

[10]:
```
df = df.dropna()
```

[11]:
```
df.shape
```

[11]:
```
(480, 12)
```

I found a reference that used the dtype of object to determine the potental fields for dummy values, I like using this method but you see in this case where I havent spilt the data set I then had to drop the target variable prior to creating the dummy variables

[12]:
```
col_cat = df.select_dtypes('object')
```

[13]:
```
col_cat.head()
```

[13]:
```
  Gender Married Dependents     Education Self_Employed Property_Area  \
1   Male     Yes          1      Graduate            No         Rural
2   Male     Yes          0      Graduate           Yes         Urban
3   Male     Yes          0  Not Graduate            No         Urban
4   Male      No          0      Graduate            No         Urban
5   Male     Yes          2      Graduate           Yes         Urban

  Loan_Status
1           N
2           Y
3           Y
4           Y
5           Y
```

[14]:
```
col_cat.drop('Loan_Status',axis=1,inplace=True)
```

[15]:
```
col_cat.head()
```

[15]:
```
  Gender Married Dependents     Education Self_Employed Property_Area
1   Male     Yes          1      Graduate            No         Rural
2   Male     Yes          0      Graduate           Yes         Urban
3   Male     Yes          0  Not Graduate            No         Urban
4   Male      No          0      Graduate            No         Urban
5   Male     Yes          2      Graduate           Yes         Urban
```

[16]:
```
df_dummies = pd.get_dummies(df[col_cat.columns])
```

[17]:
```
df = df.drop(df[col_cat.columns],axis = 1)
```

[18]:
```
df = df.join(df_dummies)
```

[19]: 
```
df.shape
```

[19]: (480, 21)

3. Split the data into a training and test set, where the "Loan_Status" column is the target.

[21]: 
```
df.head()
```

[21]:
```
   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0
5             5417             4196.0       267.0             360.0

   Credit_History Loan_Status  Gender_Female  Gender_Male  Married_No  \
1             1.0           N          False         True       False
2             1.0           Y          False         True       False
3             1.0           Y          False         True       False
4             1.0           Y          False         True        True
5             1.0           Y          False         True       False

   Married_Yes  …  Dependents_1  Dependents_2  Dependents_3+  \
1         True  …          True         False          False
2         True  …         False         False          False
3         True  …         False         False          False
4        False  …         False         False          False
5         True  …         False          True          False

   Education_Graduate  Education_Not Graduate  Self_Employed_No  \
1                True                   False              True
2                True                   False             False
3               False                    True              True
4                True                   False              True
5                True                   False             False

   Self_Employed_Yes  Property_Area_Rural  Property_Area_Semiurban  \
1              False                 True                    False
2               True                False                    False
3              False                False                    False
4              False                False                    False
5               True                False                    False

   Property_Area_Urban
1                False
2                 True
3                 True
4                 True
```

```
5                  True

[5 rows x 21 columns]
```

[22]:
```
y = df['Loan_Status']
```

[23]:
```
x = df.drop('Loan_Status',axis=1)
```

[ ]:

[24]:
```
from sklearn.model_selection import train_test_split
```

[25]:
```
x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.2)
```

[ ]:

4. Create a pipeline with a min-max scaler and a KNN classifier (see section 15.3 in the Machine Learning with Python Cookbook).

[29]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
```

[31]:
```
standardizer = StandardScaler()
```

[33]:
```
knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
```

[ ]:

[36]:
```
pipe = Pipeline([
("standardizer",standardizer),
('scaler', MinMaxScaler()),
('knn',knn)
])
```

[ ]:

[ ]:

[ ]:

[ ]:

5. Fit a default KNN classifier to the data with this pipeline. Report the model accuracy on the test set. Note: Fitting a pipeline model works just like fitting a regular model.

[43]:
```
from sklearn.metrics import accuracy_score
```

[45]:
```
knn_pipe = pipe.fit(x_train, y_train)
```

[47]:
```
y_pred = knn_pipe.predict(x_test)
```

[49]:
```
accuracy_score(y_test, y_pred)
```

[49]: 0.6875

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```

6. Create a search space for your KNN classifier where your "n_neighbors" parameter varies from 1 to 10. (see section 15.3 in the Machine Learning with Python Cookbook).

[55]:
```
search_space = [{"knn__n_neighbors": [1,2,3,4,5,6,7,8,9,10]}]
```

[ ]:
```
```

[ ]:
```
```

7. Fit a grid search with your pipeline, search space, and 5-fold cross-validation to find the best value for the "n_neighbors" parameter.

[ ]:
```
```

[ ]:
```
```

[62]:
```
classifier = GridSearchCV(pipe, search_space, cv=5, verbose=0).
 ↪fit(x_train,y_train)
```

[63]:
```
best = classifier.best_estimator_.get_params()['knn__n_neighbors']
```

[64]:
```
best
```

[64]: 7

[ ]:
```
```

8. Find the accuracy of the grid search best model on the test set. Note: It is possible that this will not be an improvement over the default model, but likely it will be.

[70]:
```
classifier.best_score_
```

[70]: 0.7265892002734108

[72]:
```
y_pred = classifier.predict(x_test)
```

Doing some research on this seciton I found that the .score gives the same result as the accuracy score and saves from having to generate the prediction variable

[77]:
```
accuracy_score(y_test, y_pred)
```

[77]:
```
0.71875
```

[74]:
```
classifier.score(x_test,y_test)
```

[74]:
```
0.71875
```

9. Now, repeat steps 6 and 7 with the same pipeline, but expand your search space to include logistic regression and random forest models with the hyperparameter values in section 12.3 of the Machine Learning with Python Cookbook.

[79]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

[81]:
```
pipe = Pipeline([("classifier", RandomForestClassifier())])
```

[83]:
```
search_space = [{"classifier":␣
↪[LogisticRegression(max_iter=500)]},{"classifier":␣
↪[RandomForestClassifier()]}]
```

[85]:
```
classifier = GridSearchCV(pipe, search_space, cv=5, verbose=0).
↪fit(x_train,y_train)
```

[86]:
```
gridsearch = GridSearchCV(pipe, search_space,cv=5,verbose=0)
```

[87]:
```
best_model = gridsearch.fit(x_train,y_train)
```

[90]:
```
print(best_model.best_estimator_)
```

```
Pipeline(steps=[('classifier', LogisticRegression(max_iter=500))])
```

[ ]:
```
```

10. What are the best model and hyperparameters found in the grid search? Find the accuracy of this model on the test set.

[95]:
```
print(best_model.best_estimator_.get_params()["classifier"])
```

```
LogisticRegression(max_iter=500)
```

[97]:
```
best_model.predict(x_test)
```

[97]:
```
array(['N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
```

```
'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
'N', 'Y', 'Y', 'Y', 'Y'], dtype=object)
```

[99]:
```
best_model.score(x_test,y_test)
```

[99]:
```
0.7395833333333334
```

[ ]:
```

```

11. Summarize your results.

This was a good use of tuning. The inital model had 68.75% accurate. The best model using only knn using had an accuracy of 71.875% showing an improvment on the original. Lastly the best model using both random forest and linear regression showed the best accuracy score of 73.95%. I feel this shows the importants of the model you use because it can greatly change the accuracy of the model.

[ ]:
```

```