Best Model for Predicting Loan Status (Use Hyperparameter Tuning)

This project predicts whether a loan will be approved. After dropping the ID column, removing incomplete rows, and one-hot–encoding categorical features, I compared three classifiers inside a scikit-learn pipeline: K-Nearest Neighbors, Logistic Regression, and Random Forest. Hyper-parameter grids and five-fold cross-validation. Logistic Regression had the best accuracy with about 80%.

In [1]:
```python
# Import packages
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.metrics import accuracy_score
```

In [2]:
```python
# Read CSV
data = pd.read_csv('/Users/Malloryh5/Downloads/Loan_Train.csv')
```

In [3]:
```python
# Check File
data.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	U
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	U
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	U
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	U

In [4]:
```python
# Drop Loan_ID
data = data.drop(columns='Loan_ID')
```

In [5]:
```python
# Drop rows with missing data
data = data.dropna(axis=0)
```

In [6]:
```python
# Convert categorical variables into dummy variables
data = pd.get_dummies(data, drop_first=True)
```

In [7]:
```python
# Check Data
data.head()
```

Out[7]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Edu
1	4583	1508.0	128.0	360.0	1.0	True	True	True	False	False	
2	3000	0.0	66.0	360.0	1.0	True	True	False	False	False	
3	2583	2358.0	120.0	360.0	1.0	True	True	False	False	False	
4	6000	0.0	141.0	360.0	1.0	True	False	False	False	False	
5	5417	4196.0	267.0	360.0	1.0	True	True	False	True	False	

In [8]:
```python
# dependant and independant variables
X = data.drop(columns="Loan_Status_Y")
y = data['Loan_Status_Y']
```

In [9]:
```python
# Split data into test data
X_test, X_train, y_test, y_train = train_test_split(X,y, test_size=.2, random_state=42)
```

In [10]:
```python
# Allow  min max standardizer
min_max = MinMaxScaler()
```

In [11]:
```python
# Allow knn classifier
knn = KNeighborsClassifier(n_neighbors=10, n_jobs=-1)
```

In [12]:
```python
# Create a pipeline with min_max scaler and knn
pipe = Pipeline([('Scaler', min_max), ('knn', knn)])
```

In [13]:
```python
# Fit the pipline to train data
print(pipe.fit(X_train, y_train))
```

```
Pipeline(steps=[('Scaler', MinMaxScaler()),
                ('knn', KNeighborsClassifier(n_jobs=-1, n_neighbors=10))])
```

In [14]:
```python
# predict y
y_pred_knn = pipe.predict(X_test)
```

In [15]:
```python
# Accuracy
print('KNN Accuracy: ', round(accuracy_score(y_test, y_pred_knn),3)*100, "%")
```

```
KNN Accuracy:  72.39999999999999 %
```

In [16]:
```python
# Create search space for knn. set n_neighbors to range 1-10.
search_sp = {'knn__n_neighbors': list(range(1,11))}
```

In [17]:
```python
# Fit grid search to pipeline
fit_grid_pipe = GridSearchCV(pipe, search_sp, cv=5, verbose=0).fit(X_train,y_train)
```

In [18]:
```
# Print best fit
print(fit_grid_pipe.best_estimator_)
print(fit_grid_pipe.best_params_)
```

```
Pipeline(steps=[('Scaler', MinMaxScaler()),
                ('knn', KNeighborsClassifier(n_jobs=-1, n_neighbors=8))])
{'knn__n_neighbors': 8}
```

In [19]:
```
# y_prdict with best fit model
y_best_pred = fit_grid_pipe.predict(X_train)
```

In [20]:
```
#Accuracy of best fit
print('KNN Best Fit Accuracy: ', round(accuracy_score(y_train, y_best_pred),3)*100, "%")
```

```
KNN Best Fit Accuracy:  74.0 %
```

In [21]:
```
# Allow logistic Regression
logistic_r = LogisticRegression()
```

In [22]:
```
# Create pipeline
pipe_random_f = Pipeline([
    ('scaler', MinMaxScaler()),
    ('classifier', KNeighborsClassifier())
                   ])
```

In [23]:
```
# Seach space for random forest and logistic regression
search_sp_lrf = [
    {'classifier': [KNeighborsClassifier()],
     'classifier__n_neighbors': list(range(1,11))
    },
    {'classifier': [LogisticRegression(max_iter=500, solver='liblinear')],
     'classifier__penalty':['l1', 'l2'],
     'classifier__C': np.logspace(0,4,10)
    },
    {'classifier': [RandomForestClassifier()],
     'classifier__n_estimators': [10,100,1000],
     'classifier__max_features':[1,2,3]
    }]
```

In [24]:
```
# Make grid search for logistic regression and random forest
grid_s_m = GridSearchCV(pipe_random_f, search_sp_lrf, cv=5, verbose=0)
```

In [25]:
```
# fit the model
best_model = grid_s_m.fit(X_train,y_train)
```

In [26]:
```
# print results
print(best_model.best_estimator_)
print(best_model.best_params_)
```

```
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('classifier',
                 LogisticRegression(max_iter=500, penalty='l1',
                                    solver='liblinear'))])
{'classifier': LogisticRegression(max_iter=500, penalty='l1', solver='liblinear'), 'classifier__C': 1.0, 'classifier__penalty': 'l1'}
```

In [27]:
```
# Find accuracy
# best model test
best_model_test = best_model.best_estimator_
```

In [28]:
```
# predict y
y_pred_m = best_model_test.predict(X_test)
```

In [29]:
```
# Accuracy score
print('Accuracy Score: ', round(accuracy_score(y_test, y_pred_m),3)*100,"%")
```

```
Accuracy Score:  80.2 %
```

KNN Classifier Accuracy: The KNN accuracy was the lowest at 72.4%.
KNN Model Accuracy: Although better than the KNN classifier, the KNN model accuracy score was still is not bad, but type grid search to see if there is a better model.
Model and Hyperparameters Accuracy: Of all three models, Logistic Regression was the best model to use. The classifier C is 1, which means it is a good, not overfitted, model. The accuracy score is over 6% higher than the KNN model at 80.2%.