Loan Approval Best Model Selection

May 12, 2024

[60]:
```
# Load

import pandas as pd
import numpy as np
file_path = '/Users/nickblackford/Desktop/Python/Loan_Train.csv'
df = pd.read_csv(file_path)
```

[61]:
```
df.head()
```

[61]:

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

[62]:
```
# Drop Loan_ID column
df = df.drop('Loan_ID', axis=1)

# Drop rows with nulls
df = df.dropna()
```

[63]:
```
# Convert the categorical features into dummy variables
df = pd.get_dummies(df, drop_first=True)
```

[64]:
```
df.head()
```

[64]:

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

[65]:
```
# Split data into training and test set
from sklearn.model_selection import train_test_split

# Split the data into features and target
X = df.drop('Loan_Status_Y', axis=1)
y = df['Loan_Status_Y']
```

[66]:
```
 # Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[67]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
```

```
from sklearn.metrics import accuracy_score

# Create a pipeline with Min-Max Scaler and KNN classifier
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('knn', KNeighborsClassifier())
])

# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[67]: 0.78125

[68]:
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for KNN
param_grid = {
    'knn__n_neighbors': range(1, 11)
}

# Create the grid search
grid_search_knn = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Fit grid search
grid_search_knn.fit(X_train, y_train)

# Best parameters and best score
best_params_knn = grid_search_knn.best_params_
best_score_knn = grid_search_knn.best_score_

best_params_knn, best_score_knn
```

[68]: ({'knn__n_neighbors': 3}, 0.7423103212576898)

[71]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Adjust the pipeline to be neutral about the model it uses
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
```

```
    ('model', KNeighborsClassifier())  # Placeholder, which will be replaced by␣
 ↪the grid search
])

# Create dictionary with corrected hyperparameters
param_grid_new = [
    {"model": [LogisticRegression(max_iter=500, solver='liblinear')],
     "model__penalty": ['l1', 'l2'],
     "model__C": np.logspace(0, 4, 10)},
    {"model": [RandomForestClassifier()],
     "model__n_estimators": [10, 100, 1000],
     "model__max_features": [1, 2, 3]}
]

# Create the grid search
grid_search_new = GridSearchCV(pipeline, param_grid_new, cv=5,␣
 ↪scoring='accuracy')

# Fit grid search
grid_search_new.fit(X_train, y_train)

# Best parameters and best score
best_params = grid_search_new.best_params_
best_score = grid_search_new.best_score_

best_params, best_score
```

[71]:
```
({'model': LogisticRegression(C=7.742636826811269, max_iter=500, penalty='l1',
                              solver='liblinear'),
  'model__C': 7.742636826811269,
  'model__penalty': 'l1'},
 0.8100136705399864)
```

Model Performance: The Logistic Regression model fit best with a cross-validation accuracy of 81%. With an 'l1' penalty and a higher regularization strength, this suggests that the model benefits from both a sparser solution and strong regularization, which helps in managing overfitting.

Comparison with Previous Models: The Logistic Regression model significantly outperformed the KNN model, which had an accuracy of 74%.

Practical Implications: With an accuracy of 81%, the model shows good potential for practical application in predicting loan approval status, assuming the dataset represents a real-world scenario accurately.