knn

December 18, 2025

1 Alternative Models: KNN

We have explored using logistic regression already and found that it gave us an accuracy of 86% While this is a great accuracy, we don't really know how well this model performs against other potential models. For this reason, this part will explore using a KNN to predict whether someone would be approved for a loan.

1.0.1 Setup and Goals

The goal of this part is to train a KNN classifier and compare it to our regularized logsitic regression model from part 2. We will use the same train/test split and preprocessing for fair comparison. We will primarily use accuracy to compare the models' performance.

[1]:
```
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

[2]:
```
df = pd.read_csv("data/cleaned_data.csv")
df.head(5)
```

[2]:
```
   age occupation_status  years_employed  annual_income  credit_score  \
0   40          Employed            17.2          25579           692
1   33          Employed             7.3          43087           627
2   42           Student             1.1          20840           689
3   53           Student             0.5          29147           692
4   32          Employed            12.5          63657           630

   credit_history_years  savings_assets  current_debt  defaults_on_file  \
0                   5.3             895         10820                 0
1                   3.5             169         16550                 0
2                   8.4              17          7852                 0
3                   9.8            1480         11603                 0
4                   7.2             209         12424                 0

   delinquencies_last_2yrs  derogatory_marks   product_type  \
```

```
0                       0                0    Credit Card
1                       1                0  Personal Loan
2                       0                0    Credit Card
3                       1                0    Credit Card
4                       0                0  Personal Loan

          loan_intent  loan_amount  interest_rate  debt_to_income_ratio  \
0            Business          600          17.02                 0.423
1    Home Improvement        53300          14.10                 0.384
2  Debt Consolidation         2100          18.33                 0.377
3            Business         2900          18.74                 0.398
4           Education        99600          13.92                 0.195

   loan_to_income_ratio  payment_to_income_ratio  loan_status
0                 0.023                    0.008            1
1                 1.237                    0.412            0
2                 0.101                    0.034            1
3                 0.099                    0.033            1
4                 1.565                    0.522            1
```

1.0.2 One Hot Encoding

[3]:
```
df = pd.get_dummies(df, columns = ['loan_intent', 'product_type',␣
 ↪'occupation_status'], drop_first = True)
df.columns
```

[3]:
```
Index(['age', 'years_employed', 'annual_income', 'credit_score',
       'credit_history_years', 'savings_assets', 'current_debt',
       'defaults_on_file', 'delinquencies_last_2yrs', 'derogatory_marks',
       'loan_amount', 'interest_rate', 'debt_to_income_ratio',
       'loan_to_income_ratio', 'payment_to_income_ratio', 'loan_status',
       'loan_intent_Debt Consolidation', 'loan_intent_Education',
       'loan_intent_Home Improvement', 'loan_intent_Medical',
       'loan_intent_Personal', 'product_type_Line of Credit',
       'product_type_Personal Loan', 'occupation_status_Self-Employed',
       'occupation_status_Student'],
      dtype='object')
```

1.0.3 Train/Test split

[4]:
```
X = df.drop(columns=['loan_status'])
y = df['loan_status']

X_train, X_test = X.iloc[:-10000], X.iloc[-10000:]
y_train, y_test = y.iloc[:-10000], y.iloc[-10000:]
```

1.0.4 Standardize features

[5]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

1.0.5 Find optimal k value

[6]:
```
k_values = range(3, 21, 2)
results = []

# iterate over each considered k value
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k, weights="distance")
    knn.fit(X_train_scaled, y_train)

    y_pred = knn.predict(X_test_scaled)

    acc = accuracy_score(y_test, y_pred)

    results.append((k, acc))

results_df = pd.DataFrame(results, columns=['k', 'accuracy'])
results_df.head()
```

[6]:

	k	accuracy
0	3	0.8542
1	5	0.8616
2	7	0.8664
3	9	0.8646
4	11	0.8667

[7]:
```
# extract k value with the best accuracy
best_k = int(results_df.loc[results_df['accuracy'].idxmax(), "k"])
print("optimal k value: ", k, " with accuracy: ")
```

```
optimal k value:  19  with accuracy:
```

1.0.6 Fitting final model with k=19

[8]:
```
knn = KNeighborsClassifier(n_neighbors=best_k, weights="distance")
knn.fit(X_train_scaled, y_train)

y_pred = knn.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
```

```
print("model accuracy: ", accuracy)
```

```
model accuracy:  0.8681
```

1.0.7 Takeaways

In the end we find that the test model accuracy for our KNN model (0.868) is very similar to our Logistic Regression model (0.868). This shows that there isn't any predictive gain from using a KNN model. In practice, the Logistic Regression model provides more clear and interpretable coefficients, which allows us to directly assess how each factor influences a person's loan approval odds. Since KNN is a black-box model, and instance based, it offers little insight into fetaure importance.

[ ]: