In [ ]:

```
#with overfitting
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, Ad
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
data = pd.read_csv(r'E:\ML\data\dataset.csv')

# Data Cleaning
data['bmi'].fillna(data['bmi'].median(), inplace=True)
data['smoking_status'].fillna(data['smoking_status'].mode()[0], inplace=True)

# Encoding categorical features
label_encoder = LabelEncoder()
data['gender'] = label_encoder.fit_transform(data['gender'])
data['ever_married'] = label_encoder.fit_transform(data['ever_married'])
data['work_type'] = label_encoder.fit_transform(data['work_type'])
data['Residence_type'] = label_encoder.fit_transform(data['Residence_type'])
data['smoking_status'] = label_encoder.fit_transform(data['smoking_status'])

# Feature Scaling
scaler = StandardScaler()
data[['avg_glucose_level', 'bmi']] = scaler.fit_transform(data[['avg_glucose_level'

# Splitting features and target
X = data.drop(columns=['id', 'stroke'])
y = data['stroke']

# Handling Imbalanced Data with SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split the resampled data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_

# Show the balance after SMOTE
print("After SMOTE, distribution of target variable:")
print(y_resampled.value_counts())

# Initialize models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Classifier': SVC(),
    'Linear Support Vector Classifier': LinearSVC(),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(),
    'Ridge Classifier': RidgeClassifier()
}

# Evaluate each model
for name, model in models.items():
    # Fit model
    model.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = model.predict(X_test)

    # Print metrics
    print(f"\n{name}:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Print confusion matrix for models where it's meaningful
    if name != 'Naive Bayes':  # Naive Bayes might not always be ideal for confusio
        print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

In [9]:

```
#checking for possibility of over fitting
data1 = pd.read_csv(r'E:\ML\data\dataset.csv')
df = pd.DataFrame(data1)


count_of_ones = df['stroke'].sum()

# Count the total number of entries in the column
total_entries = df['stroke'].count()

# Calculate the number of 0's
count_of_zeros = total_entries - count_of_ones
```

In [11]:

```
print(count_of_ones)
print(count_of_zeros)
```

```
783
42617
```

Reducing Overfitting of data using different k_neighbour parameters for the SMOTE alogorithm

In [15]:

```
#handling overfitting by changing k_neighbour parameter for each statecase for Logi
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
# Load dataset
data = pd.read_csv(r'E:\ML\data\dataset.csv')

# Data Cleaning
data['bmi'].fillna(data['bmi'].median(), inplace=True)
data['smoking_status'].fillna(data['smoking_status'].mode()[0], inplace=True)

# Encoding categorical features
label_encoder = LabelEncoder()
data['gender'] = label_encoder.fit_transform(data['gender'])
data['ever_married'] = label_encoder.fit_transform(data['ever_married'])
data['work_type'] = label_encoder.fit_transform(data['work_type'])
data['Residence_type'] = label_encoder.fit_transform(data['Residence_type'])
data['smoking_status'] = label_encoder.fit_transform(data['smoking_status'])

# Feature Scaling
scaler = StandardScaler()
data[['avg_glucose_level', 'bmi']] = scaler.fit_transform(data[['avg_glucose_level'

# Splitting features and target
X = data.drop(columns=['id', 'stroke'])
y = data['stroke']

# Define models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier()
}

# Define a range of k_neighbors
k_neighbors_values = [3, 5, 7, 10, 15]

# Store results
results = []

for k in k_neighbors_values:
    print(f"\nEvaluating with k_neighbors = {k}")

    # Apply SMOTE with current k_neighbors
    smote = SMOTE(k_neighbors=k, random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)

    # Split the resampled data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, t

    # Evaluate each model
    for name, model in models.items():
        # Fit model
        model.fit(X_train, y_train)

        # Predict and evaluate
        y_pred = model.predict(X_test)

        # Record results
        accuracy = accuracy_score(y_test, y_pred)
        results.append((k, name, accuracy))
        print(f"{name} Accuracy: {accuracy:.4f}")

# Print all results
print("\nSummary of results:")
for k, model_name, accuracy in results:
    print(f"k_neighbors = {k}, Model = {model_name}, Accuracy = {accuracy:.4f}")
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_2060\1311538129.py:13: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['bmi'].fillna(data['bmi'].median(), inplace=True)
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_2060\1311538129.py:14: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['smoking_status'].fillna(data['smoking_status'].mode()[0], inplace=True)
Evaluating with k_neighbors = 3
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Accuracy: 0.7732
Random Forest Accuracy: 0.9751

Evaluating with k_neighbors = 5
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Accuracy: 0.7786
Random Forest Accuracy: 0.9659

Evaluating with k_neighbors = 7
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression Accuracy: 0.7813
Random Forest Accuracy: 0.9600

Evaluating with k_neighbors = 10
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression Accuracy: 0.7854
Random Forest Accuracy: 0.9541

Evaluating with k_neighbors = 15
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression Accuracy: 0.7898
Random Forest Accuracy: 0.9460

Summary of results:
k_neighbors = 3, Model = Logistic Regression, Accuracy = 0.7732
k_neighbors = 3, Model = Random Forest, Accuracy = 0.9751
k_neighbors = 5, Model = Logistic Regression, Accuracy = 0.7786
k_neighbors = 5, Model = Random Forest, Accuracy = 0.9659
k_neighbors = 7, Model = Logistic Regression, Accuracy = 0.7813
k_neighbors = 7, Model = Random Forest, Accuracy = 0.9600
k_neighbors = 10, Model = Logistic Regression, Accuracy = 0.7854
k_neighbors = 10, Model = Random Forest, Accuracy = 0.9541
k_neighbors = 15, Model = Logistic Regression, Accuracy = 0.7898
k_neighbors = 15, Model = Random Forest, Accuracy = 0.9460
```

In [ ]:

```
#now checking for k parameters for each ml algo to find best algo and best k parame
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, Ad
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load dataset
data = pd.read_csv(r'E:\ML\data\dataset.csv')

# Data Cleaning
data['bmi'].fillna(data['bmi'].median(), inplace=True)
data['smoking_status'].fillna(data['smoking_status'].mode()[0], inplace=True)

# Encoding categorical features
label_encoder = LabelEncoder()
data['gender'] = label_encoder.fit_transform(data['gender'])
data['ever_married'] = label_encoder.fit_transform(data['ever_married'])
data['work_type'] = label_encoder.fit_transform(data['work_type'])
data['Residence_type'] = label_encoder.fit_transform(data['Residence_type'])
data['smoking_status'] = label_encoder.fit_transform(data['smoking_status'])

# Feature Scaling
scaler = StandardScaler()
data[['avg_glucose_level', 'bmi']] = scaler.fit_transform(data[['avg_glucose_level'

# Splitting features and target
X = data.drop(columns=['id', 'stroke'])
y = data['stroke']

# Define a range of k_neighbors
k_neighbors_values = [3, 5, 7, 10, 15]

# Define models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'AdaBoost': AdaBoostClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Classifier': SVC(),
    'Linear Support Vector Classifier': LinearSVC(),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(),
    'Ridge Classifier': RidgeClassifier()
}

# Loop through each k_neighbors value for K-Nearest Neighbors
for k in k_neighbors_values:
    print(f"\nEvaluating with k_neighbors = {k}")

    # Apply SMOTE with current k_neighbors
    smote = SMOTE(k_neighbors=k, random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)

    # Split the resampled data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, t

    # Evaluate each model
    for name, model in models.items():
        if name == 'K-Nearest Neighbors':
            model.set_params(n_neighbors=k)

        # Fit model
        model.fit(X_train, y_train)

        # Predict and evaluate
        y_pred = model.predict(X_test)

        # Print metrics
        print(f"\n{name} with k_neighbors = {k}:")
        print("Accuracy:", accuracy_score(y_test, y_pred))
        print("Classification Report:\n", classification_report(y_test, y_pred))

        # Print confusion matrix for models where it's meaningful
        if name != 'Naive Bayes':  # Naive Bayes might not always be ideal for conf
            print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_2060\4006223627.py:18: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['bmi'].fillna(data['bmi'].median(), inplace=True)
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_2060\4006223627.py:19: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['smoking_status'].fillna(data['smoking_status'].mode()[0], inplace=True)
Evaluating with k_neighbors = 3
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression with k_neighbors = 3:
Accuracy: 0.7731565671379128
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.74      0.77      8500
           1       0.76      0.80      0.78      8547

    accuracy                           0.77     17047
   macro avg       0.77      0.77      0.77     17047
weighted avg       0.77      0.77      0.77     17047

Confusion Matrix:
 [[6324 2176]
 [1691 6856]]

Random Forest with k_neighbors = 3:
Accuracy: 0.9748342816917933
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.96      0.97      8500
           1       0.96      0.99      0.98      8547

    accuracy                           0.97     17047
   macro avg       0.98      0.97      0.97     17047
weighted avg       0.98      0.97      0.97     17047

Confusion Matrix:
 [[8176  324]
 [ 105 8442]]

Gradient Boosting with k_neighbors = 3:
Accuracy: 0.8284742183375374
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.78      0.82      8500
           1       0.80      0.88      0.84      8547

    accuracy                           0.83     17047
   macro avg       0.83      0.83      0.83     17047
weighted avg       0.83      0.83      0.83     17047

Confusion Matrix:
 [[6608 1892]
 [1032 7515]]
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
19: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be rem
oved in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
AdaBoost with k_neighbors = 3:
Accuracy: 0.8019592890244618
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.76      0.79      8500
           1       0.78      0.84      0.81      8547

    accuracy                           0.80     17047
   macro avg       0.80      0.80      0.80     17047
weighted avg       0.80      0.80      0.80     17047

Confusion Matrix:
 [[6459 2041]
 [1335 7212]]

K-Nearest Neighbors with k_neighbors = 3:
Accuracy: 0.9550654074030621
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.91      0.95      8500
           1       0.92      1.00      0.96      8547

    accuracy                           0.96     17047
   macro avg       0.96      0.95      0.95     17047
weighted avg       0.96      0.96      0.95     17047

Confusion Matrix:
 [[7764  736]
 [  30 8517]]

Support Vector Classifier with k_neighbors = 3:
Accuracy: 0.771807356133044
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.70      0.75      8500
           1       0.74      0.84      0.79      8547

    accuracy                           0.77     17047
   macro avg       0.78      0.77      0.77     17047
weighted avg       0.78      0.77      0.77     17047

Confusion Matrix:
 [[5969 2531]
 [1359 7188]]
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\svm\_classes.py:31: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\svm\_base.py:1237: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
Linear Support Vector Classifier with k_neighbors = 3:
Accuracy: 0.7172523024579105
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.44      0.61      8500
           1       0.64      0.99      0.78      8547

    accuracy                           0.72     17047
   macro avg       0.81      0.72      0.69     17047
weighted avg       0.81      0.72      0.69     17047

Confusion Matrix:
 [[3765 4735]
 [  85 8462]]

Naive Bayes with k_neighbors = 3:
Accuracy: 0.7530943861089928
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.70      0.74      8500
           1       0.73      0.80      0.77      8547

    accuracy                           0.75     17047
   macro avg       0.76      0.75      0.75     17047
weighted avg       0.76      0.75      0.75     17047


Decision Tree with k_neighbors = 3:
Accuracy: 0.954302809878571
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      8500
           1       0.94      0.97      0.95      8547

    accuracy                           0.95     17047
   macro avg       0.95      0.95      0.95     17047
weighted avg       0.95      0.95      0.95     17047

Confusion Matrix:
 [[8006  494]
 [ 285 8262]]

Ridge Classifier with k_neighbors = 3:
Accuracy: 0.7729219217457617
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.73      0.76      8500
           1       0.75      0.81      0.78      8547

    accuracy                           0.77     17047
   macro avg       0.77      0.77      0.77     17047
weighted avg       0.77      0.77      0.77     17047
```

```
Confusion Matrix:
 [[6237 2263]
 [1608 6939]]

Evaluating with k_neighbors = 5
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression with k_neighbors = 5:
Accuracy: 0.7785534111573884
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.75      0.77      8500
           1       0.76      0.81      0.79      8547

    accuracy                           0.78     17047
   macro avg       0.78      0.78      0.78     17047
weighted avg       0.78      0.78      0.78     17047

Confusion Matrix:
 [[6377 2123]
 [1652 6895]]

Random Forest with k_neighbors = 5:
Accuracy: 0.9653898046577111
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.95      0.96      8500
           1       0.95      0.98      0.97      8547

    accuracy                           0.97     17047
   macro avg       0.97      0.97      0.97     17047
weighted avg       0.97      0.97      0.97     17047

Confusion Matrix:
 [[8046  454]
 [ 136 8411]]

Gradient Boosting with k_neighbors = 5:
Accuracy: 0.8325218513521441
Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.78      0.82      8500
           1       0.80      0.88      0.84      8547

    accuracy                           0.83     17047
   macro avg       0.84      0.83      0.83     17047
weighted avg       0.84      0.83      0.83     17047

Confusion Matrix:
 [[6650 1850]
 [1005 7542]]
```

```
E:\DE\Anaconda_ide\Anaconda\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
19: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be rem
oved in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
AdaBoost with k_neighbors = 5:
Accuracy: 0.8084706986566551
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.77      0.80      8500
           1       0.79      0.85      0.82      8547

    accuracy                           0.81     17047
   macro avg       0.81      0.81      0.81     17047
weighted avg       0.81      0.81      0.81     17047

Confusion Matrix:
 [[6542 1958]
 [1307 7240]]

K-Nearest Neighbors with k_neighbors = 5:
Accuracy: 0.9297823663987799
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.87      0.92      8500
           1       0.88      0.99      0.93      8547

    accuracy                           0.93     17047
   macro avg       0.94      0.93      0.93     17047
weighted avg       0.94      0.93      0.93     17047

Confusion Matrix:
 [[7361 1139]
 [  58 8489]]
```

In [ ]:

```
#tabulate for all K's for all algo, then find best. Publish this paper.
```