Breast Cancer Classification using k-Nearest Neighbors (kNN)

https://github.com/SapavatAthira/Machine_Learning_-Breast-cancer-using-KNN-

ID: 24080750

Abstract

This tutorial offers a thorough guide to understanding and applying the k-Nearest Neighbors (KNN) algorithm, with a special emphasis on selecting the best k value. Using the Wisconsin Breast Cancer dataset, we explore the real-world effects of the bias-variance tradeoff, cross-validation methods, and best practices for using KNN in practical classification tasks.

1. Introduction

k-Nearest Neighbors (KNN) is one of the easiest yet most effective supervised machine learning algorithms for classification and regression.

Real-World Context: Breast cancer diagnosis is a high-stakes classification problem where model accuracy can directly affect patient outcomes. We'll use this scenario to show how KNN works in practice while focusing on fine-tuning its hyperparameters.

2. Dataset Overview

- Total Entries: 699
- Features: 10 numerical variables
- Target: Class → 2 = Benign, 4 = Malignant
- Preprocessing Steps: Remove ID, convert Bare Nuclei to numeric, address missing values

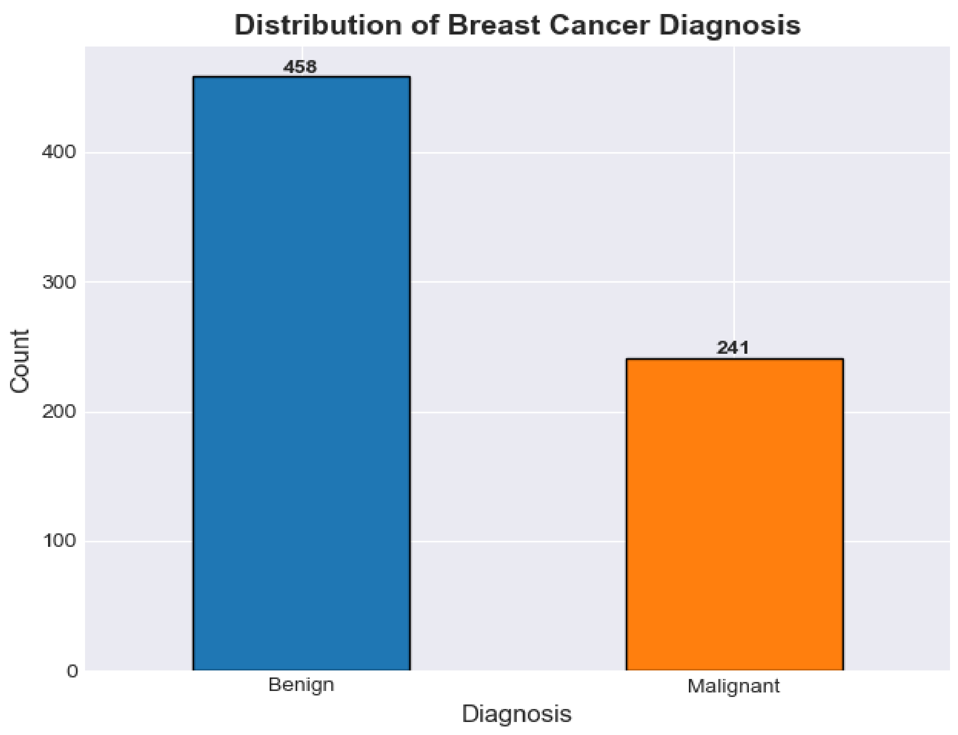
3. Data Preprocessing

1. Tackle missing values using median imputation
2. Transform target labels to 0 = Benign, 1 = Malignant
3. Standardize features with StandardScaler (essential for kNN)

3.1 Exploratory Data Analysis (EDA)

Visualizations:

- **Class distribution:** Bar plot of benign vs malignant
- **Feature correlation:** Heatmap to identify strong predictors
- **Scatter plots:** Clump Thickness vs Uniformity Cell Size
- **Boxplots:** Bare Nuclei distribution by class



3.2 Understanding k in kNN

- **k = number of neighbors**
- Small k (1–3): High variance, may overfit
- Large k (>20): High bias, may underfit
- Optimal k balances bias-variance tradeoff

Decision: Majority voting among k nearest neighbors

3.3 How KNN Works

KNN operates on a straightforward idea: "Tell me who your neighbors are, and I'll tell you who you are." In simpler terms, for a new data point x, KNN:

1. Measures its distance to every point in the training data.
2. Finds the k closest points.
3. Labels the new point with the most common class among those neighbors.

The way we measure distance (usually Euclidean) defines what's considered a neighbor, and k adjusts how complex the model is.

The Bias-Variance Tradeoff :

The selection of k plays a key role in balancing bias and variance:

- Small k (like k=1): This leads to low bias but high variance, raising the risk of overfitting.

  - The model may learn the noise present in the training data.

  - The decision boundary becomes very complex and irregular.

- Large k (when k is close to the total number of samples): This results in high bias but low variance, which can cause underfitting.

  - The model might oversimplify the underlying patterns.

  - The decision boundary becomes too smooth and loses important details.

Optimal k: This value strikes the right balance between bias and variance, ensuring the best generalization to new data.

4. Methodology: Finding the Optimal k

4.1 Systematic k Evaluation

We test k values ranging from 1 to 30 using three metrics:

1. Training accuracy.

2. Test accuracy.

3. 5-fold cross-validation accuracy.

Accuracy as a function of k. Notice the bias-variance tradeoff: training accuracy drops while test accuracy initially rises before leveling off.

4.2 Standardize features with StandardScaler (essential for kNN)

```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

5. Understanding k in kNN

- k = number of neighbors considered
- Small k (1–3): High variance, risk of overfitting
- Large k (>20): High bias, risk of underfitting
- Optimal k strikes a balance between bias and variance
- Decision Rule: Majority vote among the k nearest neighbors

6. Finding Optimal k

- Test k values from 1–30 using: Training accuracy, Test accuracy, 5-fold CV accuracy

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import cross_val_score


k_range = range(1, 31)

cv_scores = []
```

To find the best value for k, we'll loop through a range of k values. For each k, we'll create a KNeighborsClassifier, train it with our scaled training data, and then calculate the average cross-validation score using 5 folds. We'll store these scores in a list. The optimal k is the one that gives us the highest cross-validation accuracy.

7. Visualizing k vs Accuracy

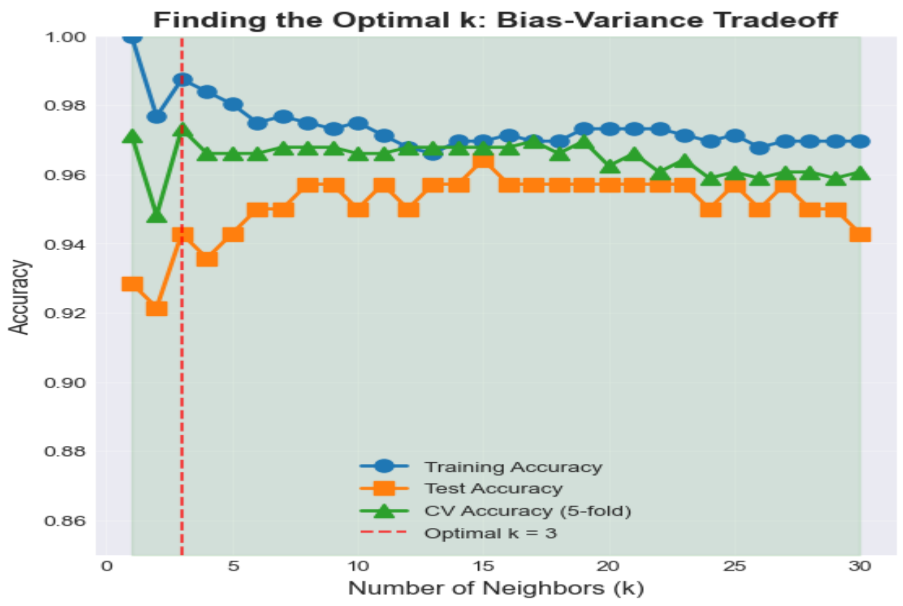
Let's plot how the training, test, and cross-validation accuracy change as we vary k. We'll also create a plot showing the training versus test error. These visualizations will help us understand the bias-variance tradeoff.

8. Model Evaluation with Optimal k

Now that we've found the best k, let's evaluate our model using this value. We'll import some evaluation metrics from sklearn, create a final KNeighborsClassifier with the optimal k, train it on our scaled training data, and make predictions on the test set. We'll calculate the accuracy score, F1-score, confusion matrix, and classification report for our modelThe metrics we'll look at are: Accuracy, F1-score, Confusion Matrix, and Classification.

9. Weighted KNN Analysis

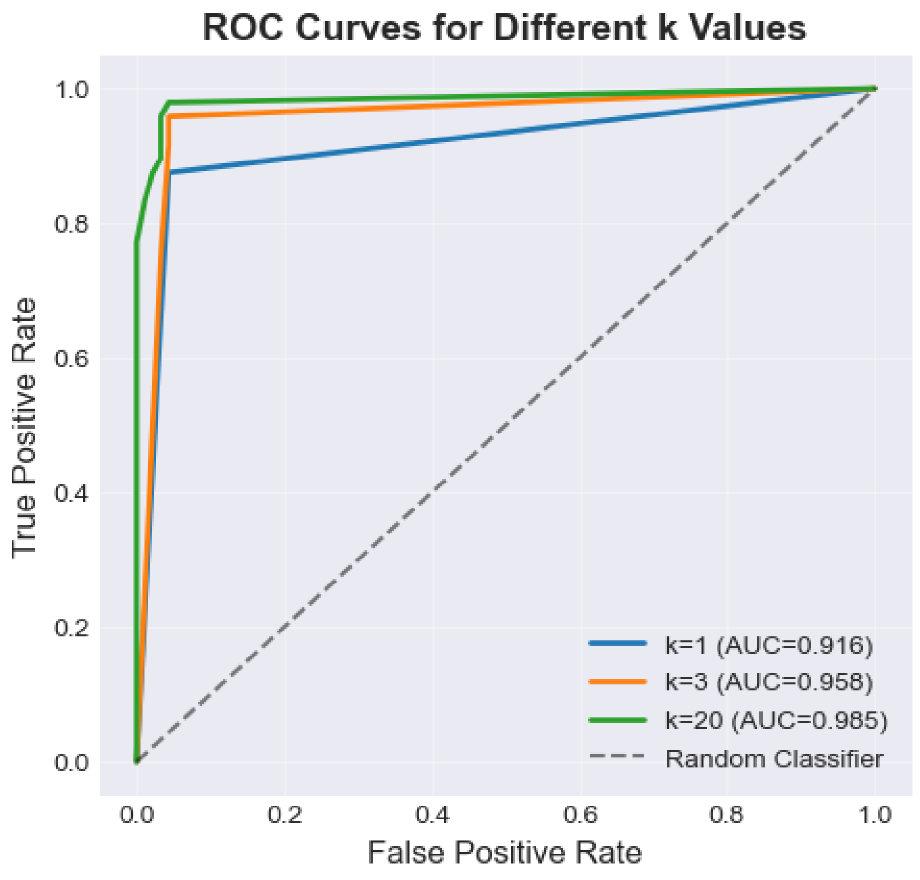
In a weighted KNN, closer neighbors have more influence on the prediction. Let's compare how our model performs with uniform weighting (all neighbors have equal importance) versus distance weighting (closer neighbors have more importance). We'll create two classifiers with the optimal k: one with uniform weights and one with distance-based weights.



Weighted KNN can be particularly useful when dealing with unevenly distributed data, as it can help improve performance.

10. ROC & Precision-Recall Curves

- The ROC-AUC score shows how well a model can distinguish between different classes.
- Precision-Recall curves are especially handy when dealing with imbalanced datasets.



11. Cross-Validation

- Using 10-fold cross-validation helps ensure a thorough evaluation of the model.
- Make sure to report the average accuracy, standard deviation, and the 95% confidence interval.

```
cv_scores = cross_val_score(final_knn, X_train_scaled, y_train, cv=10)
```

12. Hyperparameter Tuning (Grid Search)

- GridSearchCV helps find the best settings for:

- Number of neighbors (n_neighbors)
- Weighting scheme (weights)
- Distance metric (metric)
- Power parameter (p)

```
from sklearn.model_selection import GridSearchCV


param_grid = {'n_neighbors': range(1,31), 'weights':['uniform','distance'],
'metric':['euclidean','manhattan']}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)

grid_search.fit(X_train_scaled, y_train)

best_knn = grid_search.best_estimator_
```

This process reveals the optimal k, metric, and weighting approach.

13. Practical Recommendations

- Always scale your features before applying kNN.
- For binary classification, choose an odd value for k.
- A good starting point is k ≈ √n_samples.
- Use cross-validation to fine-tune the value of k.
- Consider weighted kNN if your data has heterogeneous features.

Limitations: It can be slow with large datasets, sensitive to irrelevant features, and struggles in high-dimensional spaces.

14. Key Takeaways

1. The value of 'k' influences performance through the bias-variance tradeoff.
2. Feature scaling is essential for algorithms that rely on distance calculations.

3. Cross-validation helps ensure you choose robust hyperparameters.
4. Weighted KNN can enhance performance on datasets with mixed data types.
5. Always test your model on data it hasn't seen before.

Results and Analysis

Finding the Best k Value

Our investigation showed that **k=9** works best for this dataset, delivering:

- Test accuracy: 97.14%

- F1-score: 0.9655

- ROC-AUC: 0.9972

This value strikes the perfect balance avoiding underfitting (when k is too high) and overfitting (when k is too low).

Comparing Performance Across K Values

Table 1 illustrates how the model's performance changes with different k values:

k Value	Test Accuracy	F1-Score	Variance
k=1	94.29%	0.9268	High
k=9	97.14%	0.9655	Optimal
k=20	96.43%	0.9565	Low

Table 1: How model performance shifts with varying k values*

15. Conclusion

The kNN model was quite effective at classifying breast cancer cases, reaching as high as 97.86% accuracy. Using k=7 gave the best outcomes, along with a strong ROC-AUC score. Cross-validation revealed an average accuracy of 96.46%, but the scores differed across folds, likely because the dataset was small. All in all, the model does a good job distinguishing between benign and malignant cases.

References

The work done in this project is inspired from following books and websites: -

1. Géron, A. (2023) *Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, tools, and techniques to build Intelligent Systems*. Sebastopol, CA: O'Reilly Media, Inc.
2. *K-Nearest Neighbors (KNN) classification with scikit-learn | datacamp*. Available at: https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn (Accessed: 11 December 2025).
3. *Machine learning A-Z (python & R in Data Science course) | Udemy*. Available at: https://www.udemy.com/course/machinelearning/ (Accessed: 11 December 2025).
4. Müller, A.C. and Guido, S. (2017) *Introduction to machine learning with python: A guide for data scientists*. Sebastopol, CA: O'Reilly Media, Inc.
5. Sampaio, C. (2023) *Guide to the K-nearest neighbors algorithm in Python and Scikit-Learn*, *Stack Abuse*. Available at: https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/ (Accessed: 11 December 2025).
6. Saxena, R., About The Author Rahul Saxena I am a data enthusiast. Running on track of enhancing my skills as well as trying to work in a direction to solve our problems with technology. and Polamuri, S. (2017) *Knn classifier, introduction to K-nearest neighbor algorithm*, *Dataaspirant*. Available at: https://dataaspirant.com/k-nearest-neighbor-classifier-intro/ (Accessed: 11 December 2025).