Tutorial Report: k-Nearest Neighbours (k-NN) for Classification

Student Id: 24069965

Github Link: https://github.com/prardhanmushke/knn-tutorial

1. Introduction

One of the least complex and most intuitive machine-learning algorithms is the k-Nearest Neighbours (k-NN). It works on the concept that objects of similar nature would fall into the same category. In contrast with most others, k-NN does not construct an explicit mathematical formulation of the training data. Rather, it reproduces all the data points and only predicts when a new unknown sample is presented. Due to this behaviour, it is said to be a lazy learning algorithm. Simple as it is, k-NN is astonishingly good at a variety of classification tasks especially in the cases when the data is small and well-organized. The given tutorial is meant to describe k-NN in a clear and professional way with the help of working Python code and plots in the Jupyter notebook environment.

2. Conceptual Understanding of k-NN

The intuition of k-NN is simple. As an algorithm is fed with an input sample to classify, the algorithm computes the distance between the input sample and each sample in the training sample. Euclidean distance is usually used to compute distance, but it can also be computed by other measures like Manhattan distance or Minkowski distance depending on the nature of the problem. After these distances have been computed, the algorithm finds the k nearest neighbours of the training samples, which are training samples nearest to the input point. The most common class in these neighbours defines the class that the new sample is expected to belong to. Basically, the algorithm makes a decision on the basis of a local voting mechanism.

Among the strong aspects of k-NN, one may note the fact it has very little assumptions concerning the underlying data distribution. In comparison to other algorithms, like the logistic regression or support machine, algorithms that make certain geometric or statistical assumptions, k-NN just uses the similarity measure. Nevertheless, this advantage is also a disadvantage where data of high dimensionality is concerned as distance measures lose their significance in high dimensionality a phenomenon usually called the curse of dimensionality.

Due to this reason preprocessing techniques like scaling of features and dimensionality reduction may be used to enhance the performance of k-NN.

3. The k Parameter and Its Influence

The central parameter of the algorithm performance is the parameter k that dictates the number of neighbours that affect the prediction. When k is small, then the model becomes noise sensitive thus overfitting. On the other hand, a very high value of k can over fit the decision boundaries and result in underfitting. Hence, it is important to choose a suitable value of k, and the common methods of doing so empirically are validation methods like grid search or cross-validation. In practice, odd values of k are commonly used in order to minimize the possibility of a tie among votes of the classes.

Besides the choice of k, the distance measure also influences the results of classification. Euclidean distance is common in continuous numerical variables, however where one has categorical or binary variables, different measures or data transformations are required. Euclidean distance is the most appropriate in the case of this tutorial based on the famous Iris dataset of continuous features.

4. Dataset Description

In order to illustrate the application of k-NN, this tutorial makes use of the Iris data which is openly accessible at the UCI Machine learning Repository. The data set has 150 samples of iris flowers in three species: Setosa, Versicolor and Virginica. There are four numerical characteristics of each sample, including sepal length, sepal width, petal length, and petal width. The Iris dataset is more often than not used in introductory machine-learning experiments, due to its small scale and the fact that classes are well-separated. The tutorial makes the dataset used directly available on an online source, meaning that it makes it reproducible in Google Colab or any other cloud-based Jupyter environment without uploading files.

5. Preprocessing and Normalisation

Normalisation of the dataset must be done before k-NN is applied as the distance-based algorithms are sensitive to the features scale. Aspects of measurement on larger number scales can prevail in the distance computations unless they are appropriately normalised. Here, scikit-learn has been used to achieve standardisation with its StandardScaler that scales the mean of

each feature to zero and scales the variance of each feature to one. Besides, a common 80-20 division is employed to divide the dataset into training and testing parts, so that the performance of the model might be tested accordingly.

6. Implementing k-NN in Python

With the scikit-learn library, the k-NN can be easily implemented. The training model is then trained on the training data with the KNeighborsClassifier which has been preprocessed. The training data are stored in the internal memory of the classifier and the search of neighbours is done only on the occasion of prediction. In order to assess the model, accuracy is calculated on the test set and a confusion plot is plotted to visualise misclassifications. Moreover, a decision-boundary plot is also provided by projecting the data to two main components using PCA which enables us to see how the k-NN divides the classes in a 2-dimensional feature space.

CODE SNIPPETS

A. Data Loading and Preprocessing:

```
# Load dataset
url = "https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv"
df = pd.read_csv(url)

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

B. k-NN Model Training:

```
 # Initialize and train k-NN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

C. Evaluation:

```
# Predict and evaluate
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

7. Performance Evaluation

The k-NN works well in general on the Iris dataset. Accuracy and visual decision boundaries reflect the success of the technique since the dataset is small and the separation between classes

is quite good. In the confusion matrix, there is usually very slight misclassification between Versicolor and Virginica, the two species that are the nearest to each other in the feature space. This visual evidence helps to improve the comprehension and give a better understanding of the behaviour of neighbourhood-based decision boundaries.

8. Conclusion

K-Nearest neighbours is a basic but effective classification algorithm which is applicable to a huge number of basic machine-learning problems. This tutorial presented the theoretical basis of the method and gave an end-to-end and reproducible implementation in Python of a publicly available dataset. Since k-NN assumes very few conditions, it is the best to teach the principles of supervised learning and emphasize on the key concepts of distance measures, decision limits and how feature scaling affects performance. The code and notebook they are accompanied with are handy to explore the algorithm and set parameters like k to see the effect they have on the performance of a model. The simplicity, transparency and clarity of k-NN has seen it as a useful method in introductory teaching of machine-learning.

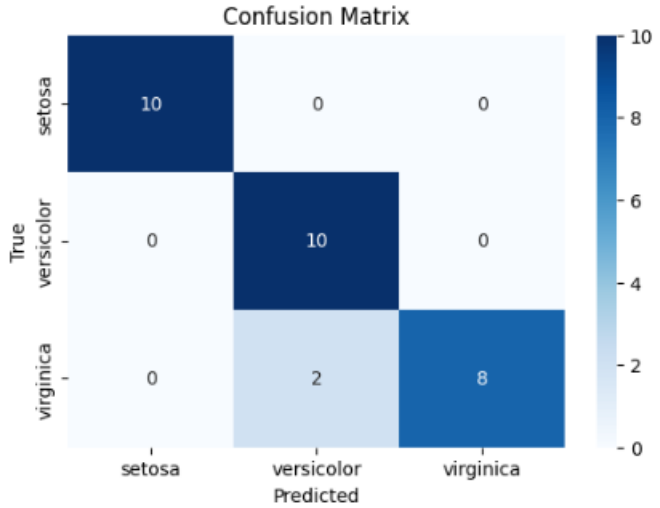
Python Results

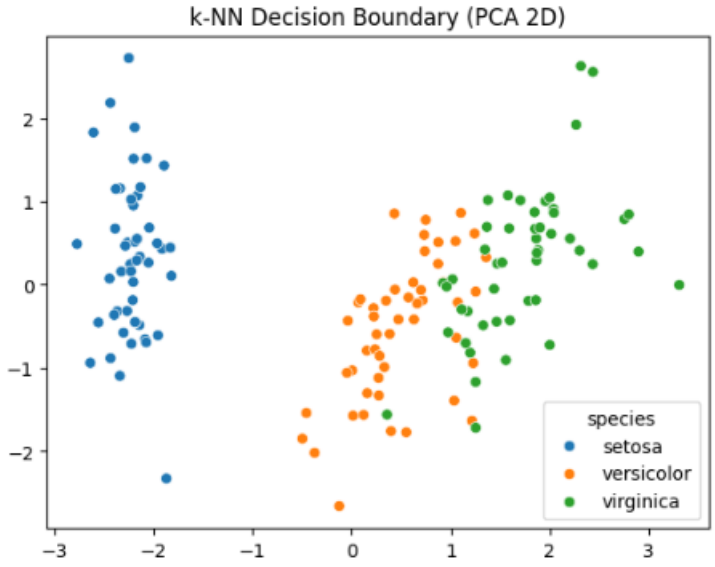
```
Dataset Loaded Successfully
Accuracy: 0.9333333333333333

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.83      1.00      0.91        10
   virginica       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30
```





Analysis and Critical Evaluation of k-NN Model Results

The overall accuracy of the k-Nearest Neighbours classifier on the Iris dataset was 93.33 which shows that it has a strong predictive performance. Nevertheless, further analysis of the classification report indicates that as good as the model is overall, it has minor differences in the accuracy with which it identifies the three species. Critical analysis of precision, recall and f1-scores offers a perspective of the classifier behaviour in underlining and pinpointing its strengths and weaknesses.

All the samples of Iris Setosa were correctly classified by the model with precision of 1.00, recall of 1.00 and f1-score of 1.00. This is in agreement with the fact that Setosa is separable in the Iris dataset. Setosa differs uniquely with the other two species in the petal measurements, thus it is the easiest species to differentiate. It is not surprising that the model has worked flawlessly on Setosa and this is a validation of the ability of k-NN to identify the obvious distance-based boundaries.

In the case of Iris Versicolor, the model got a precision of 0.83 and a recall of 1.00. This would indicate that all of the real Versicolor were properly determined, whereas some of the samples of Virginica were determined as Versicolor. The false negative is zero based on the recall of 1.00 but the precision is low which implies that there were false positives. This discrepancy is

indicative of the fact that the Versicolor and Virginica are closer in terms of feature space, and consequently, their bounds are softer and more affected by the k selection as well as feature scaling.

The same was not the case with Virginica: it had a flawless precision (1.00) but a recall of 0.80. Practically, this implies that, whenever a specific classifier gave Virginica an answer, it was giving the correct answer, but it was not able to recognise two Virginica samples and misclassified them as Versicolor. This confirms that Virginica is more overlapping with Versicolor than with Setosa and it indicates that the neighbourhood of the two wrongly classified points was more dominated by Versicolor samples, which may be because there was similarity in features or noise.

The macro average f1-score of 0.93 and weighted average of 0.93 indicate that the performance was balanced across the classes although there are variations between classes. As the dataset is balanced across the classes, weighting does not drastically change the averages, although, it supports the idea that the classifier is still performing well on the overall dataset.

Despite the good accuracy and classification measures, the assessment should be able to critically reflect the weaknesses of k-NN. The algorithm is an oracle of the value of k chosen; k=5 had solid results, but varying values could have varying results. A smaller k may overfit and capture noise to give irregular decision boundaries whereas a larger k may over smooth. The cross-validation might be useful in selecting the best k more accurately.

Also, being a distance-based algorithm, k-NN requires the scaling of features. The model was enhanced by standardisation though further progress may be possible by first experimenting with MinMax scaling or PCA prior to classification. The misclassifications can be seen as the PCA decision-boundary plot, where Versicolor and Virginica overlap to a large extent when it is reduced to two components.

Overall, it can be concluded that although the k-NN classifier shows high predictive power on the Iris dataset with more than 93% accuracy, a critical analysis has provided the variations in performance of the classifier on the specific classes, specifically Versicolor and Virginica. This model is sound and can be improved by hyperparameter tuning, other scaling methods and dimensionality reduction.

References

[1] Scikit-learn: Machine Learning in Python, Pedregosa et al., Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[2] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7(2), 179-188.

[3] Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21-27.

[4] UCI Machine Learning Repository: Iris Dataset. Retrieved from https://archive.ics.uci.edu/ml/datasets/iris

[5] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences.

[6] scikit-learn documentation: k-Nearest Neighbors. Retrieved from https://scikit-learn.org/stable/modules/neighbors.html