Lecture 15

KNN Classification vs. Regression

Differences Between KNN for Classification vs. Regression:

Objective:

- Classification: The objective of KNN in classification is to assign a class label to a new data point based on the majority class among its nearest neighbors. The output is a discrete class label.
- Regression: The objective of KNN in regression is to predict a continuous value for a new data point based on the average (or weighted average) of the values of its nearest neighbors. The output is a continuous numerical value.

Decision Rule:

- Classification: The decision is based on the mode (most common value) among the nearest neighbors. If there's a tie, various tiebreaking methods can be employed, such as random selection, or using a weighted vote.
- Regression: The decision is based on averaging the target values of the nearest neighbors. This can be a simple arithmetic mean or a weighted mean where closer neighbors have a higher influence.

Metrics:

- Classification: Performance metrics include accuracy, precision, recall, F1-score, confusion matrix, ROC curves, and AUC.
- Regression: Performance metrics include mean squared error (MSE), mean absolute error (MAE), root mean squared error (RMSE), R-squared, and mean absolute percentage error (MAPE).

Output Type:

- Classification: Outputs are categorical.
- Regression: Outputs are continuous numerical values.

Pros and Cons of Using KNN for Regression:

Pros:

Simplicity: KNN is straightforward to understand and implement, making it accessible for those new to machine learning. It requires minimal assumptions about the underlying data distribution.

Non-parametric: KNN is a non-parametric method, meaning it doesn't assume a specific form for the underlying data distribution or the relationship between the input and output variables. This makes it flexible and capable of modeling complex relationships.

Adaptability: KNN can adapt to different types of data and can be easily customized by choosing different distance metrics, weighting schemes, and the number of neighbors (k).

Local Approximation: KNN effectively captures local patterns in data, which can be beneficial when the relationship between variables varies in different parts of the feature space.

Cons:

Computational Cost: KNN can be computationally expensive, especially with large datasets. The algorithm needs to calculate distances to every point in the dataset for each prediction, which can be slow, especially when k is large.

Curse of Dimensionality: In high-dimensional spaces, data points tend to become equidistant from each other, making distance-based methods like KNN less effective. The performance of KNN can degrade significantly as the number of features increases.

Sensitivity to Noise: KNN can be sensitive to noisy data and outliers because each neighbor has an equal influence on the prediction. This can lead to poor predictions if the nearest neighbors are not representative of the actual trend.

Choice of k: The performance of KNN is highly dependent on the choice of the parameter k. A small k can lead to overfitting, while a large k can lead to underfitting. There is no universal rule for choosing the best k, so it often requires tuning.

Lack of Model Interpretability: Unlike linear regression, KNN does not provide an interpretable model. It does not give insights into the relationships between the features and the target variable, as it simply memorizes the training data.

Boundary Effects: KNN can struggle near the boundaries of the data distribution, where there may be fewer neighbors to average. This can lead to less reliable predictions in those regions.

You can use other methods to make the predictions from the nearest neighbors. The typical arithmetic mean is the most common. However, there are several other methods that can be used to make predictions in KNN regression. Here are some alternatives:

Weighted Mean:
- Description: Instead of giving each neighbor equal weight, you can give more weight to closer neighbors and less weight to farther ones. The weights are typically inversely related to the distance between the point and its neighbors.
- Common Weighting Schemes:
  - **Inverse Distance Weighting**: The weight is the inverse of the distance (e.g., $weight_i = \frac{1}{d_i}$, where d_i is the distance to the ith neighbor).
  - **Gaussian Kernel Weighting**: The weight is computed using a Gaussian (or other kernel) function, where closer neighbors have exponentially more influence.
- Advantages: This approach can make predictions more robust, especially when neighbors are unevenly distributed in the feature space.

Median of Neighbors:
- Description: Instead of taking the mean, you take the median of the target values of the k-nearest neighbors. This can be useful in cases where the target variable contains outliers.
- Advantages: The median is less sensitive to outliers than the mean, making it a good choice in noisy datasets.

Mode (for integer target variables):

- Description: If the target variable is a count or an integer value, the mode (most common value) among the k-nearest neighbors can be used.
- Advantages: The mode is useful when you want to predict the most frequently occurring integer value, especially when your target variable is discrete but numeric.
- Disadvantages: sometimes R has issues when calculated a mode for a numeric value, so this can present coding challenges, particularly if there is more than one mode, or if there is no mode.

Trimmed Mean:

- Description: This involves calculating the mean after discarding a certain percentage of the smallest and largest values among the neighbors. For example, a 10% trimmed mean would discard the top and bottom 10% of values and take the mean of the remaining 80%.
- Advantages: This can help reduce the influence of extreme outliers on the prediction.

Harmonic Mean:

- Description: The harmonic mean is another way to average the values, which is particularly useful if the values you're predicting are rates or ratios.
- Advantages: The harmonic mean gives less weight to large values, which can be beneficial in certain types of datasets.

Quantile Prediction:

- Description: Instead of predicting the mean or median, you could predict a specific quantile (e.g., the 25th or 75th percentile) of the target values of the k-nearest neighbors.
- Advantages: Quantile regression can be useful when you are interested in predicting the range or distribution of the target variable, rather than just a central tendency.

Local Polynomial Regression:

- Description: Instead of taking a simple mean, you could fit a local polynomial regression (e.g., locally weighted scatterplot smoothing or LOWESS) to the k-nearest neighbors and use this to make the prediction.
- Advantages: This method can capture more complex, non-linear relationships between the neighbors.

Let’s compare the standard KNN classification algorithm to the standard KNN regression algorithm. We’ll start with the classification example.

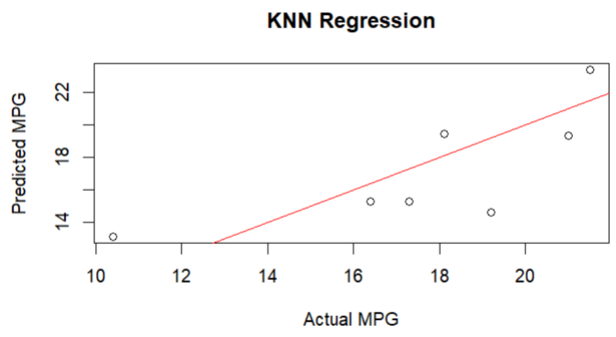
```
# Load necessary libraries
library(datasets)

# Load the Iris dataset
data(iris)

# Prepare the data by removing the Species column to separate features and labels
iris_data <- iris[, -5]
iris_labels <- iris[, 5]

# Euclidean distance function
euclidean_distance <- function(x, y) {
  return(sqrt(sum((x - y)^2)))
```

```r
}

# KNN classification function
knn_classify <- function(train_data, train_labels, test_data, k = 3) {
  predictions <- c()

  for (i in 1:nrow(test_data)) {
    # Calculate distances from the test point to all training points
    distances <- apply(train_data, 1, function(row) euclidean_distance(row, test_data[i, ]))

    # Sort the distances and get the indices of the k nearest neighbors
    k_nearest_indices <- order(distances)[1:k]

    # Get the labels of the k nearest neighbors
    k_nearest_labels <- train_labels[k_nearest_indices]

    # Determine the most common label (mode) among the k nearest neighbors
    predicted_label <- names(sort(table(k_nearest_labels), decreasing = TRUE))[1]

    # Append the prediction to the list of predictions
    predictions <- c(predictions, predicted_label)
  }

  return(predictions)
}

# Split the data into training and testing sets (80/20 split)
set.seed(123) # for reproducibility
train_indices <- sample(1:nrow(iris_data), size = 0.8 * nrow(iris_data))

train_data <- iris_data[train_indices, ]
train_labels <- iris_labels[train_indices]

test_data <- iris_data[-train_indices, ]
test_labels <- iris_labels[-train_indices]

# Run the KNN classifier with k=3
k <- 3
predictions <- knn_classify(train_data, train_labels, test_data, k = k)

# Calculate accuracy
accuracy <- sum(predictions == test_labels) / length(test_labels)
print(paste("Accuracy:", accuracy))

# Confusion matrix
table(Predicted = predictions, Actual = test_labels)
```

Now, the regression case.

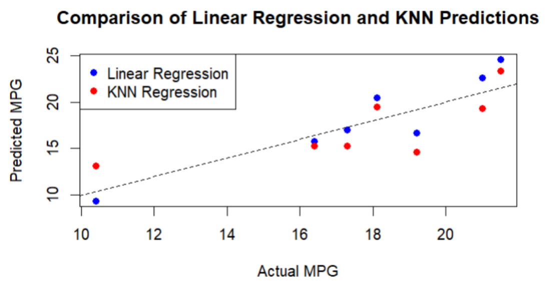
```r
# Load the mtcars dataset
data(mtcars)

# Prepare the data by selecting features and the target variable
mtcars_data <- mtcars[, c("disp", "hp", "wt")]
mtcars_labels <- mtcars[, "mpg"]

# Euclidean distance function (same as above)
euclidean_distance <- function(x, y) {
  return(sqrt(sum((x - y)^2)))
}

# KNN regression function
knn_regress <- function(train_data, train_labels, test_data, k = 3) {
  predictions <- c()

  for (i in 1:nrow(test_data)) {
    # Calculate distances from the test point to all training points
    distances <- apply(train_data, 1, function(row) euclidean_distance(row, test_data[i, ]))

    # Sort the distances and get the indices of the k nearest neighbors
    k_nearest_indices <- order(distances)[1:k]

    # Get the labels (mpg values) of the k nearest neighbors
    k_nearest_labels <- train_labels[k_nearest_indices]

    # Predict the mean of the k nearest labels
    predicted_value <- mean(k_nearest_labels)

    # Append the prediction to the list of predictions
    predictions <- c(predictions, predicted_value)
  }

  return(predictions)
}

# Split the data into training and testing sets (80/20 split)
set.seed(123) # for reproducibility
train_indices <- sample(1:nrow(mtcars_data), size = 0.8 * nrow(mtcars_data))

train_data <- mtcars_data[train_indices, ]
train_labels <- mtcars_labels[train_indices]

test_data <- mtcars_data[-train_indices, ]
test_labels <- mtcars_labels[-train_indices]

# Run the KNN regressor with k=3
k <- 3
```

```r
predictions <- knn_regress(train_data, train_labels, test_data, k = k)

# Calculate Mean Absolute Error (MAE)
mae <- mean(abs(predictions - test_labels))
print(paste("Mean Absolute Error (MAE):", mae))

# Plot predictions vs actual values
plot(test_labels, predictions, xlab = "Actual MPG", ylab = "Predicted MPG", main = "KNN
Regression")
abline(0, 1, col = "red") # Reference line
```



One way that we can compare the predictions of linear regression to the results of KNN regression is to plot the original points and the predictions for both on the same graph.

```r
# Step 1: Fit a Linear Regression Model
linear_model <- lm(mpg ~ disp + hp + wt, data = mtcars)

# Step 2: Generate Predictions from the Linear Regression Model
linear_predictions <- predict(linear_model, newdata = test_data)

# Step 3: Generate Predictions from the KNN Regression Model (using k=3 as before)
k <- 3
knn_predictions <- knn_regress(train_data, train_labels, test_data, k = k)

# Step 4: Create a Comparison Plot
plot(test_labels, linear_predictions, col = "blue", pch = 16,
    xlab = "Actual MPG", ylab = "Predicted MPG",
    main = "Comparison of Linear Regression and KNN Predictions")
points(test_labels, knn_predictions, col = "red", pch = 16)

# Add a legend
legend("topleft", legend = c("Linear Regression", "KNN Regression"),
     col = c("blue", "red"), pch = 16)

# Add a reference line for perfect prediction
abline(0, 1, col = "black", lty = 2)
```



You can see that the predictions are often similar, but not always on the same side of the perfect-fit line.

We'll see as we look at more classification methods that use this voting strategy that we used in KNN that they can also be modified to perform regression in similar ways. We'll look at decision trees next, and we'll include an example or two in that discussion of the regression vs. classification treatment in that model as well, particularly when we look at ensemble methods.

Resources:

1. https://towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955