Lecture 14

KNN, tie-breaking methods

In k-Nearest Neighbors (KNN) classification, ties can occur when multiple classes have the same number of nearest neighbors. This is particularly likely when kkk is even, but it can happen in other cases as well. There are several common methods for breaking ties in KNN:

1. Random Selection

- *Description:* If two or more classes are tied for the most neighbors, a random choice is made between the tied classes.
- *Pros:* Simple to implement.
- *Cons:* Adds a non-deterministic element to the classification, which can lead to different results on different runs.

2. Choose the Class of the Closest Neighbor

- *Description:* In the event of a tie, choose the class of the closest individual neighbor among the tied classes.
- *Pros:* Leverages the most immediate information about the distance.
- *Cons:* May disproportionately favor the class of the closest point, even if other neighbors suggest a different class.

3. Choose the Class with the Smallest Sum of Distances

- *Description:* Sum the distances of the neighbors for each class, and choose the class with the smallest sum.
- *Pros:* Accounts for the distribution of the neighbors' distances in the decision-making process.
- *Cons:* Requires additional computation and might not always be intuitively the best choice.

4. Choose the Class with the Highest Prior Probability

- *Description:* If there's a tie, choose the class with the highest prior probability (based on the class distribution in the training data).
- *Pros:* Incorporates knowledge of class distribution.
- *Cons:* Might introduce bias toward more frequent classes, potentially overlooking the true neighborhood distribution.

5. Weighted Voting

- *Description:* Instead of counting neighbors equally, weight them by the inverse of their distance, so closer neighbors have more influence. If this still results in a tie, one of the above methods can be applied.
- *Pros:* Intuitive approach that reduces the likelihood of ties.
- *Cons:* More complex to implement and compute.

6. Use a Weighted Majority Vote (Different Weights)

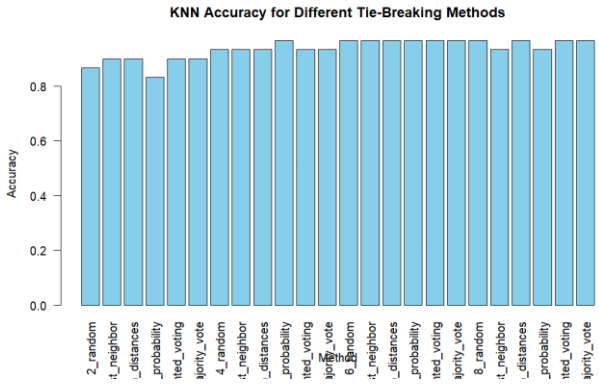
- *Description:* In a tie, weights based on additional criteria (such as feature importance or confidence scores) can be used to break the tie.
- *Pros:* Provides flexibility in handling ties with additional information.
- *Cons:* Requires defining and computing additional weights, which may complicate the algorithm.

Let's look at an example that applies various tie-breaking methods. We'll remove one of the species from the iris dataset to encourage more ties to occur. Then we'll look at what happens with the results from different tiebreaking methods. The best way to assess the stability of the random method is to run it many times and see how it changes with each run.

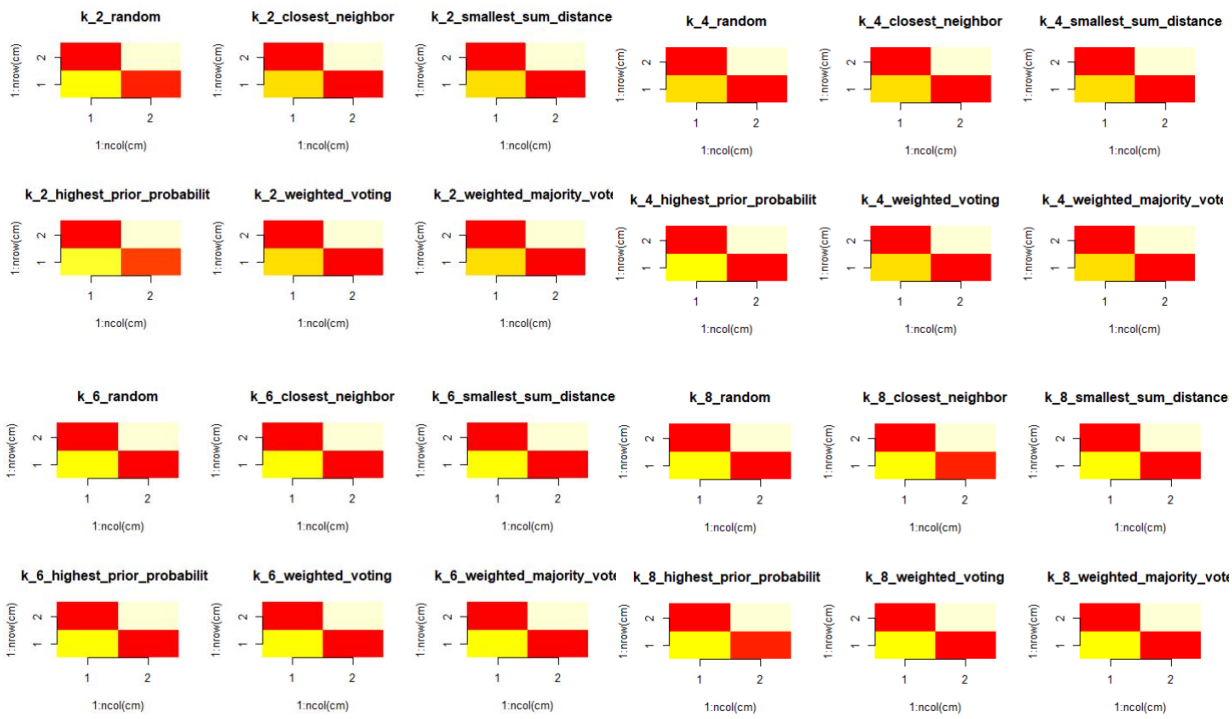
```
# Load the iris dataset
data(iris)

# Remove the setosa species
iris <- subset(iris, Species != "setosa")

# Encode the Species column as numeric
iris$Species <- as.numeric(factor(iris$Species))

# Split the data into training and test sets (70-30 split)
set.seed(42)
train_index <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[train_index, ]
test_data <- iris[-train_index, ]

# Helper function to calculate Euclidean distance
euclidean_distance <- function(x, y) {
  sqrt(sum((x - y) ^ 2))
}

# Implement KNN with tie-breaking methods
knn_tie_break <- function(train_data, test_data, k, tie_breaking_method) {
  predictions <- c()

  for (i in 1:nrow(test_data)) {
    distances <- apply(train_data[, -5], 1, euclidean_distance, test_data[i, -5])
    nearest_neighbors <- order(distances)[1:k]
    neighbor_classes <- train_data[nearest_neighbors, 5]
    neighbor_distances <- distances[nearest_neighbors]

    # Count frequencies of each class in nearest neighbors
    class_counts <- table(neighbor_classes)
    max_count <- max(class_counts)
    tied_classes <- as.integer(names(class_counts[class_counts == max_count]))

    # Apply tie-breaking method
    if (length(tied_classes) > 1) {
      if (tie_breaking_method == "random") {
        selected_class <- sample(tied_classes, 1)

      } else if (tie_breaking_method == "closest_neighbor") {
        closest_class <- neighbor_classes[which.min(neighbor_distances)]
        selected_class <- ifelse(closest_class %in% tied_classes, closest_class, tied_classes[1])
```

```
    } else if (tie_breaking_method == "smallest_sum_distances") {
      sum_distances <- sapply(tied_classes, function(cls) {
        sum(neighbor_distances[neighbor_classes == cls])
      })
      selected_class <- tied_classes[which.min(sum_distances)]

    } else if (tie_breaking_method == "highest_prior_probability") {
      # Assume equal prior probability for this case, so pick the first tied class
      selected_class <- tied_classes[1]

    } else if (tie_breaking_method == "weighted_voting") {
      weighted_vote <- sapply(tied_classes, function(cls) {
        sum(1 / neighbor_distances[neighbor_classes == cls])
      })
      selected_class <- tied_classes[which.max(weighted_vote)]

    } else if (tie_breaking_method == "weighted_majority_vote") {
      weighted_vote <- sapply(tied_classes, function(cls) {
        sum(1 / neighbor_distances[neighbor_classes == cls])
      })
      selected_class <- tied_classes[which.max(weighted_vote)]
    }
   } else {
     selected_class <- tied_classes
   }

   predictions <- c(predictions, selected_class)
  }

  return(predictions)
}

# Function to evaluate the KNN classifier
evaluate_knn <- function(predictions, true_labels) {
  accuracy <- sum(predictions == true_labels) / length(true_labels)
  cm <- table(Predicted = predictions, Actual = true_labels)
  return(list(accuracy = accuracy, confusion_matrix = cm))
}

# Evaluate different tie-breaking methods
tie_break_methods <- c("random", "closest_neighbor", "smallest_sum_distances",
                       "highest_prior_probability", "weighted_voting", "weighted_majority_vote")

results <- list()

for (k in c(2, 4, 6, 8)) {
  for (method in tie_break_methods) {
```

```
    predictions <- knn_tie_break(train_data, test_data, k, method)
    evaluation <- evaluate_knn(predictions, test_data$Species)
    results[[paste0("k_", k, "_", method)]] <- evaluation
  }
}

# Plot accuracy for each tie-breaking method
accuracy_results <- sapply(results, function(res) res$accuracy)
names(accuracy_results) <- gsub("k_", "k = ", names(accuracy_results))

barplot(accuracy_results, las=2, col="skyblue", main="KNN Accuracy for Different Tie-Breaking
Methods",
        ylab="Accuracy", xlab="Method")

# Visualize confusion matrices
par(mfrow = c(2, 3))
for (i in 1:length(results)) {
  cm <- results[[i]]$confusion_matrix
  image(1:ncol(cm), 1:nrow(cm), t(as.matrix(cm)), col = heat.colors(12), axes = FALSE)
  axis(1, at = 1:ncol(cm), labels = colnames(cm))
  axis(2, at = 1:nrow(cm), labels = rownames(cm))
  title(main = names(results)[i])
}
```



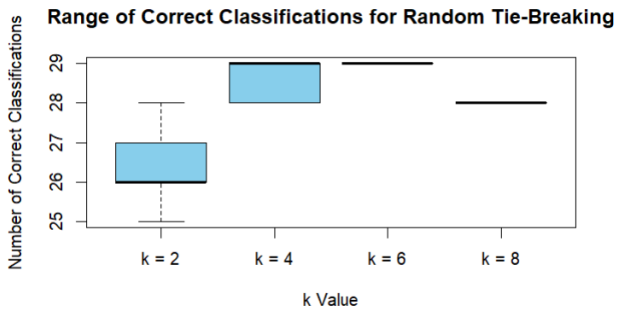
The labels are cut off on this graph, but the six methods are run in order from k=2 on the left, through k=8 on the right, with each new level starting with random. Below we see plots of the confusion matrices.



Let's dig a little more deeply into the stability of using the random tiebreaker on the iris dataset. Bear in mind that the more overlap there is between classes (and there is some here), the more likely you will be to encounter a need to break ties. When there are clear gaps between the classes, this will tend to come up less often.

```
# Load necessary libraries
library(class)

# Load the iris dataset
data(iris)

# Remove the setosa species
iris <- subset(iris, Species != "setosa")

# Encode the Species column as numeric (1 = versicolor, 2 = virginica)
iris$Species <- as.numeric(factor(iris$Species))

# Split the data into training and test sets (70-30 split)
set.seed(42)
train_index <- sample(1:nrow(iris), 0.7 * nrow(iris))
train_data <- iris[train_index, ]
test_data <- iris[-train_index, ]

# Separate predictors and labels
train_predictors <- train_data[, -5]
train_labels <- train_data[, 5]
test_predictors <- test_data[, -5]
```

```
test_labels <- test_data[, 5]

# Function to run KNN with random tie-breaking and record correct classifications
run_knn_simulation <- function(k, train_predictors, train_labels, test_predictors, test_labels,
n_simulations = 500) {
  correct_classifications <- numeric(n_simulations)

  for (i in 1:n_simulations) {
    set.seed(i)  # Set seed for reproducibility
    predictions <- knn(train_predictors, test_predictors, train_labels, k = k)
    correct_classifications[i] <- sum(predictions == test_labels)
  }

  return(correct_classifications)
}

# Run simulations for different values of k
set.seed(42)
k_values <- c(2, 4, 6, 8)
simulation_results <- lapply(k_values, run_knn_simulation, train_predictors, train_labels,
test_predictors, test_labels)
names(simulation_results) <- paste("k =", k_values)

# Analyze the range, mean, and variance of correct classifications
results_summary <- lapply(simulation_results, function(x) {
  list(
    min_correct = min(x),
    max_correct = max(x),
    mean_correct = mean(x),
    sd_correct = sd(x)
  )
})

# Print results summary
results_summary

# Visualize the range of correct classifications
boxplot(simulation_results,
        main = "Range of Correct Classifications for Random Tie-Breaking",
        xlab = "k Value", ylab = "Number of Correct Classifications",
        col = "skyblue")
```



As we can see from the boxplots, smaller values of k generate more ties (in this case) than larger values, and the range of cases is therefore larger and more subject to a range of outcomes.

In addition to the six tiebreaking methods we've looked at already, there are a few others that could be used.

Lexicographical Order:
When a tie occurs, the classes are ranked alphabetically (or numerically) and the first class in lexicographical order is chosen. This method is often used as a deterministic fallback when other methods do not resolve the tie.

Distance-weighted KNN:
Similar to weighted voting, but instead of weighting by the number of votes, the weights are inversely proportional to the distance. The closer a neighbor, the higher its influence on the prediction. If a tie occurs, it can be resolved by picking the class with the highest cumulative weight.

Fractional Votes:
Each class receives a fractional vote based on how close it is to the test point relative to other classes. For example, if a class is twice as close as another class, it receives twice the weight. The tie can then be broken by choosing the class with the highest fractional vote.

Kernel Density Estimation (KDE):
KDE can be used to estimate the probability density of each class in the neighborhood. The class with the highest estimated density is chosen. This method often results in smoother decision boundaries and can naturally resolve ties.

Custom Heuristics:
In some domains, domain-specific knowledge might suggest a custom heuristic for breaking ties. For example, in a medical diagnosis system, ties might be broken in favor of the more severe diagnosis to ensure caution in treatment.

Probabilistic Method:
Assign probabilities to each class based on the distribution of classes in the neighborhood. The tie can be broken by sampling from the probability distribution, where the class with the highest probability is more likely to be chosen.

For the last example, we'll look at the kernel density tiebreaker.

```r
# Load necessary libraries
library(MASS)  # For kde2d function
library(datasets)  # Iris dataset

# Set a seed for reproducibility (other than 42)
set.seed(123)

# Load the Iris dataset and remove Setosa species
data(iris)
iris <- iris[iris$Species != "setosa", ]

# Function to calculate Euclidean distance
euclidean_distance <- function(x1, x2) {
  sqrt(sum((x1 - x2)^2))
}

# KDE function for tiebreaking
kde_tiebreak <- function(neighbors, neighbor_classes, new_point) {
  # Estimate density for each class
  class_levels <- unique(neighbor_classes)
  densities <- sapply(class_levels, function(cls) {
    cls_points <- neighbors[neighbor_classes == cls, ]
    if (nrow(cls_points) > 1) {
      kde <- kde2d(cls_points[, 1], cls_points[, 2], n = 50)
      estimate <- approx(kde$x, kde$y, new_point[1])$y
      return(estimate)
    } else {
      # Return a small density if there's only one point
      return(1e-10)
    }
  })

  # Return the class with the highest estimated density
  return(class_levels[which.max(densities)])
}

# KNN classification function with KDE tiebreaking
knn_kde_tiebreak <- function(train_data, train_labels, test_point, k) {
  # Calculate distances to all training points
  distances <- apply(train_data, 1, euclidean_distance, x2 = test_point)

  # Get the indices of the k-nearest neighbors
  k_nearest_idx <- order(distances)[1:k]

  # Get the classes of the k-nearest neighbors
  k_nearest_classes <- train_labels[k_nearest_idx]

  # If there's a tie, break it using KDE
```

```
  unique_classes <- unique(k_nearest_classes)
  if (length(unique_classes) > 1) {
    # Use the nearest neighbors' features for KDE
    neighbors <- train_data[k_nearest_idx, ]
    return(kde_tiebreak(neighbors, k_nearest_classes, test_point))
  } else {
    # No tie, return the majority class
    return(unique_classes[1])
  }
}

# Split the dataset into training and testing sets
train_indices <- sample(1:nrow(iris), size = 0.8 * nrow(iris))
train_data <- iris[train_indices, -5]
train_labels <- iris[train_indices, 5]
test_data <- iris[-train_indices, -5]
test_labels <- iris[-train_indices, 5]

# Apply the KNN with KDE tiebreaking to the test set
predictions <- sapply(1:nrow(test_data), function(i) {
  knn_kde_tiebreak(train_data, train_labels, test_data[i, ], k = 2)
})

# Calculate accuracy
accuracy <- mean(predictions == test_labels)
cat("Accuracy:", accuracy, "\n")

# Confusion Matrix
confusion_matrix <- table(Predicted = predictions, Actual = test_labels)
print(confusion_matrix)
```

One issue I noticed is that setosa is still included in the confusion matrix, even though it was removed as an option from the dataset. Despite that, all the other observations in the test set were correctly classified. 100% accuracy is hard to come by.

Resources:

1. https://www.linkedin.com/pulse/breaking-ties-k-nn-classification-nicholas-pylypiw/
2. https://medium.com/@kirudang/a-caveat-of-k-nearest-neighbors-knn-a-tie-scenario-87cd96f0780f