Analiza funkcji **knn** dla różnych wywołań oraz w kontekście innych funkcji klasyfikacyjnych

Emil Dragańczuk

17 maj 2020

Wprowadzenie

W poniższym dokumencie przeanalizujemy funkcję `knn` względem poszczególnych jej wywołań oraz na tle lasów losowych `randomForset::randomForest()`. Będziemy rozważać 3 zbiory danych: `abalone.csv`, `kinematics.csv` oraz `winequality-red.csv`. Benchmarki bazują na pięciokrotnej kroswalidacji i zostały już wykonane oraz zapisane w analogicznych plikach csv, aby poniższy dokument budował się krócej niż 10 godzin. Do eksportowania benchmarków zostały użyte poniższe funkcje:

```
shortenKs <- function(ks, num) {
  t(apply(ks, 1, head, n = num))
}

createKnnBenchmark <- function(fileName) {
  dataSet <- abalone <- read.csv(paste("TestData\\", fileName, sep = ""))

  aggregators <- c(moda, srednia_a, mediana, minkara1.5, minkara3.0, wazonyrand)
  aggNames <- c("moda", "srednia_a", "mediana", "minkara1.5", "minkara3.0", "wazonyrand")
  ks <- seq(1, 19, 2)
  maxk <- max(ks)
  ps <- c(1, 2, Inf)
  results <- list()
  titles <- vector()
  params <- matrix(nrow = 0, ncol = 3)
  result <- matrix(nrow = 0, ncol = 3)

  n <- nrow(dataSet)
  sampledData <- dataSet[sample(n, size = n, replace = FALSE),]
  labels <- sampledData[,1]
  variables <- data.matrix(sampledData[, -1])

  for (i in 0:4) {
    results[[i+1]] <- matrix(nrow = 0, ncol = 3)

    start <- floor(i / 5 * n) + 1
    end <- floor((i + 1) / 5 * n)
    len <- end - start

    trainingSet <- variables[-end:-start,]
    trainingLabels <- labels[-end:-start]
    testSet <- variables[start:end,]
    correctLabels <- labels[start:end]
```

```
    for (p in ps) {
      maxkLabels <- knn(trainingSet, trainingLabels, testSet, maxk, p)

      for (k in ks) {
        kLabels <- shortenKs(maxkLabels, k)

        for (aggId in 1:6) {
          resultLabels <- aggregators[[aggId]](kLabels)

          err <- sum(resultLabels != correctLabels) / len / 5
          mad <- sum(abs(resultLabels - correctLabels)) / len / 5
          mse <- sum((resultLabels - correctLabels) ^ 2) / len / 5

          results[[i+1]] <- rbind(results[[i+1]], c(err, mad, mse))

          if (i==0) {
            params <- rbind(params, c(p, k, aggId))
            titles <- append(titles, paste("knn dla p=", p, ", k=", k, ", agregator ", aggNames[aggId],
          }
        }
      }
    }

  }

  for (i in 1:5) {
    if (i == 1) {
      result <- results[[i]]
    }
    else {
      result <- result + results[[i]]
    }
  }

  result <- t(apply(result, 1, function(r) c(round((r[1] * 100), 3), round(r[2], 4), round(r[3], 4))))

  result <- cbind(result, params)
  colnames(result) <- c("err", "mad", "mse", "p", "k", "aggId")
  rownames(result) <- titles

  write.csv(result, file = fileName)
}
```

Analiza efektywności funkcji `knn` względem jej poszczególnych parametrów

W tej podsekcji zajmiemy się analizą funkcji `knn` bazując na wyliczonych już benchmarkach. Rozważymy błędy w zależności od pojedynczych parametrów k, p oraz agregatorów, uśredniając błędy przy agregacji.

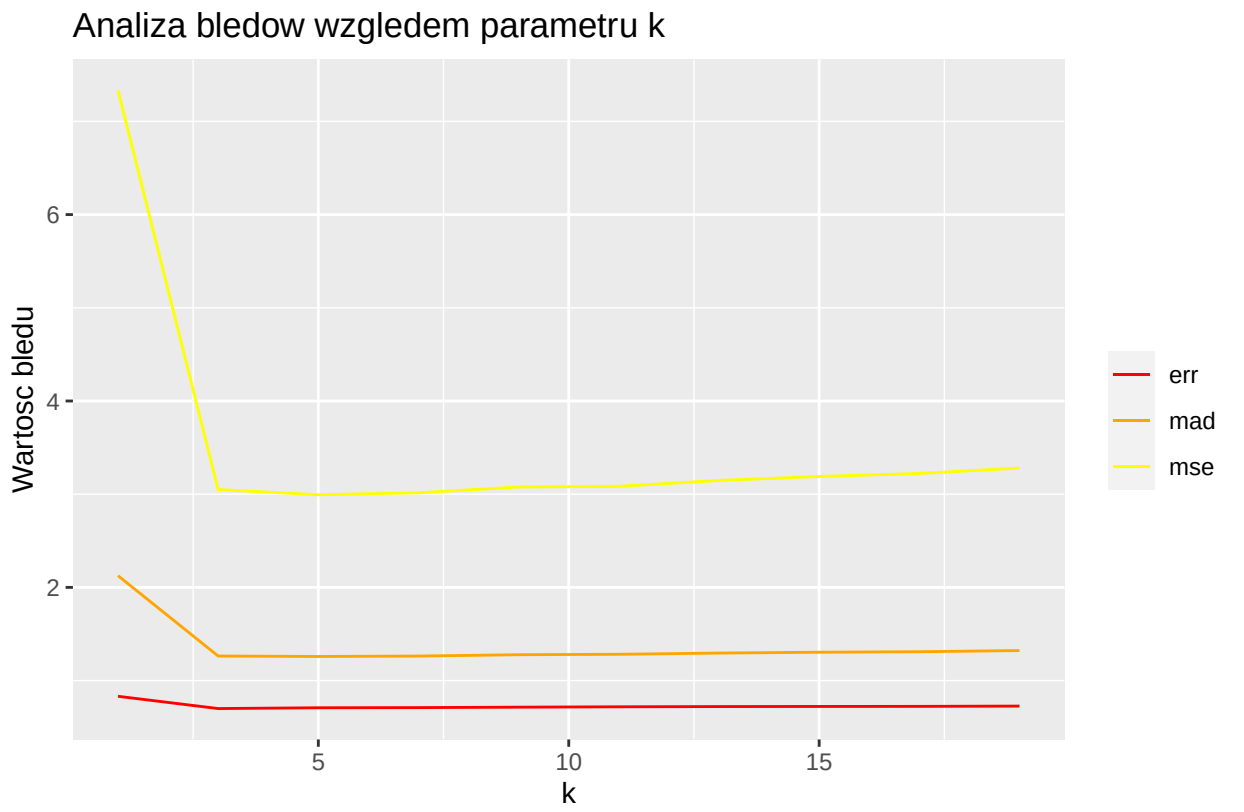
Wpływ parametru k

Do generowania wykresów błędów względem ilości najbliższych sąsiadów użyjemy poniższej funkcji:

```
plotKErrs <- function(dataSet) {
  aggregatedSet <- group_by(dataSet, k) %>%
    summarise(err = mean(err/100), mad = mean(mad), mse = mean(mse))

  ggplot(aggregatedSet, aes(x=k)) +
    geom_line(aes(y=err, colour = "err")) +
    geom_line(aes(y=mad, colour = "mad")) +
    geom_line(aes(y=mse, colour = "mse")) +
    labs(title = "Analiza bledow wzgledem parametru k", y="Wartosc bledu", x="k", caption="") +
    scale_colour_manual("",
                        breaks = c("err", "mad", "mse"),
                        values = c("red", "orange", "yellow"))
}
```

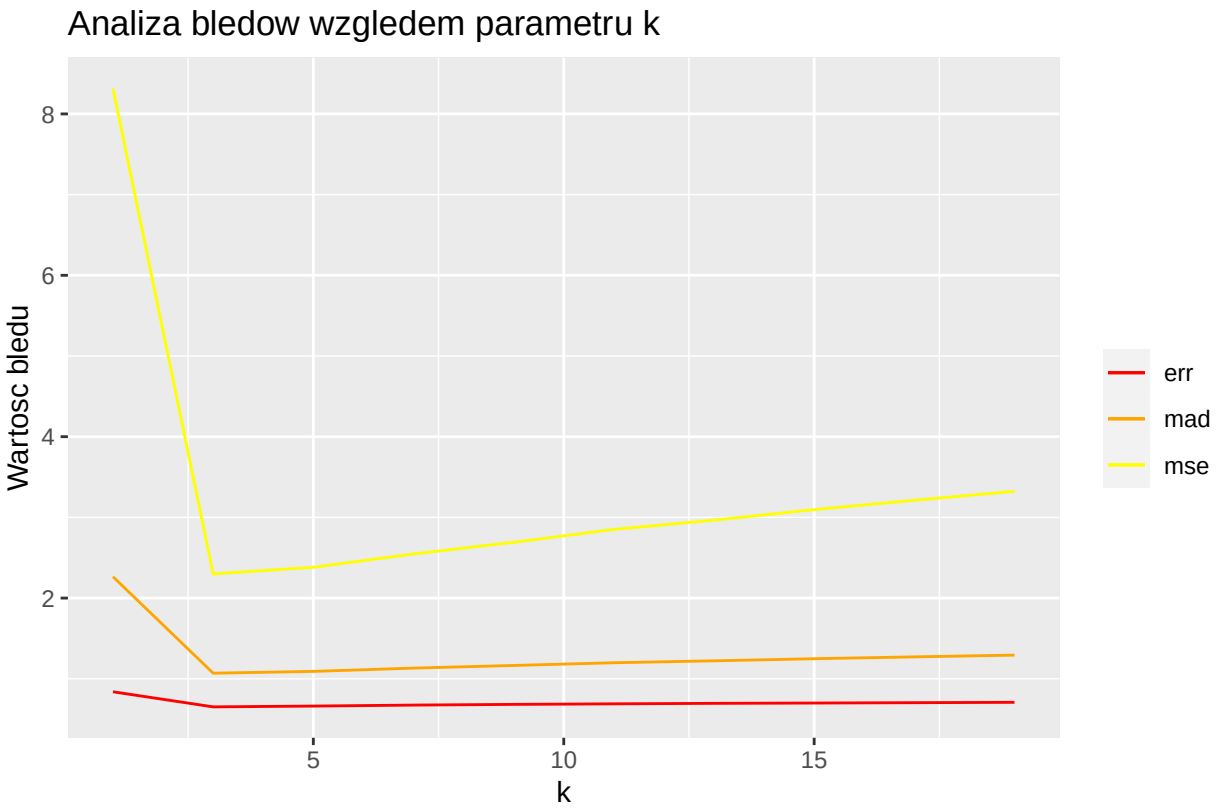
Dane `abalone.csv`

```
plotKErrs(abaloneKnnBenchmarks)
```



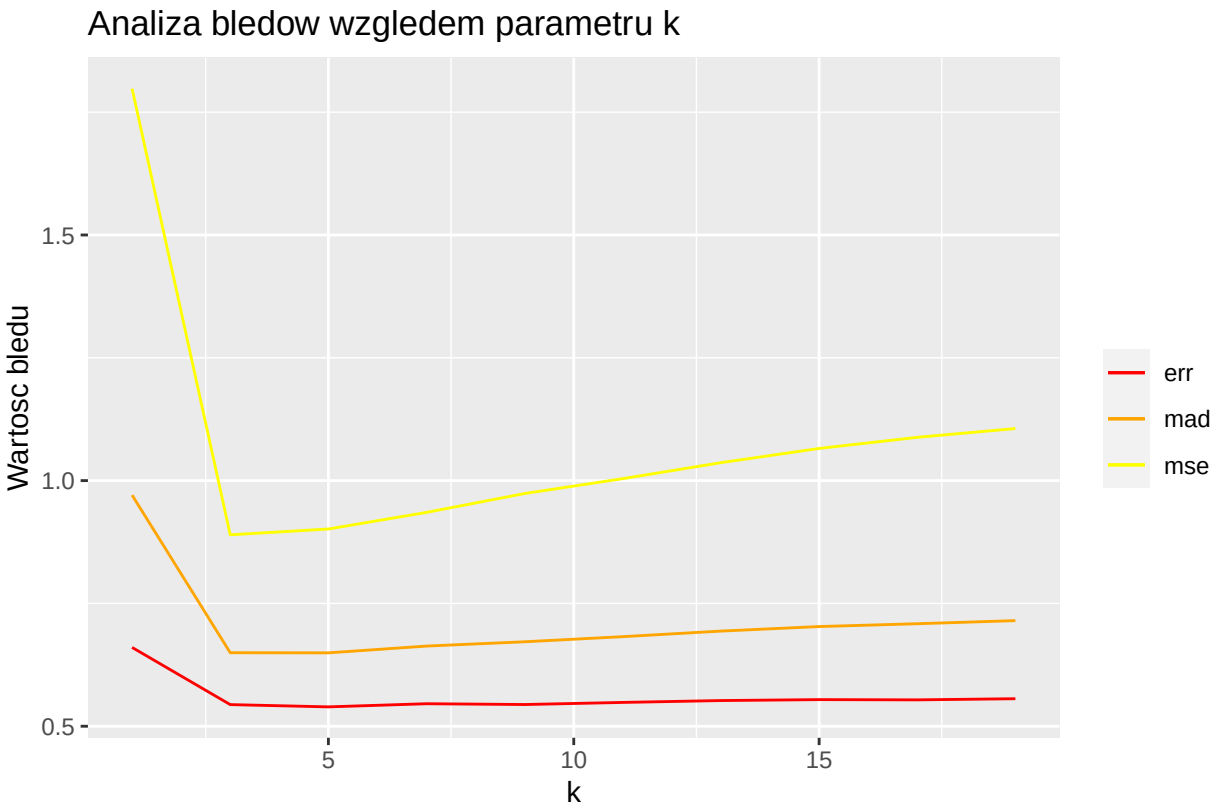
Dane `kinematics.csv`

```
plotKErrs(kinematicsKnnBenchmarks)
```



Dane `winequality-red.csv`

```
plotKErrs(redwineKnnBenchmarks)
```



Wnioski

Można zauważyć że gdy bierzemy tylko jedną najbliższą wartość, błędy się znacznie podwyższają. Lecz od $k = 3$ widzimy powolny, liniowy przyrost błędów.

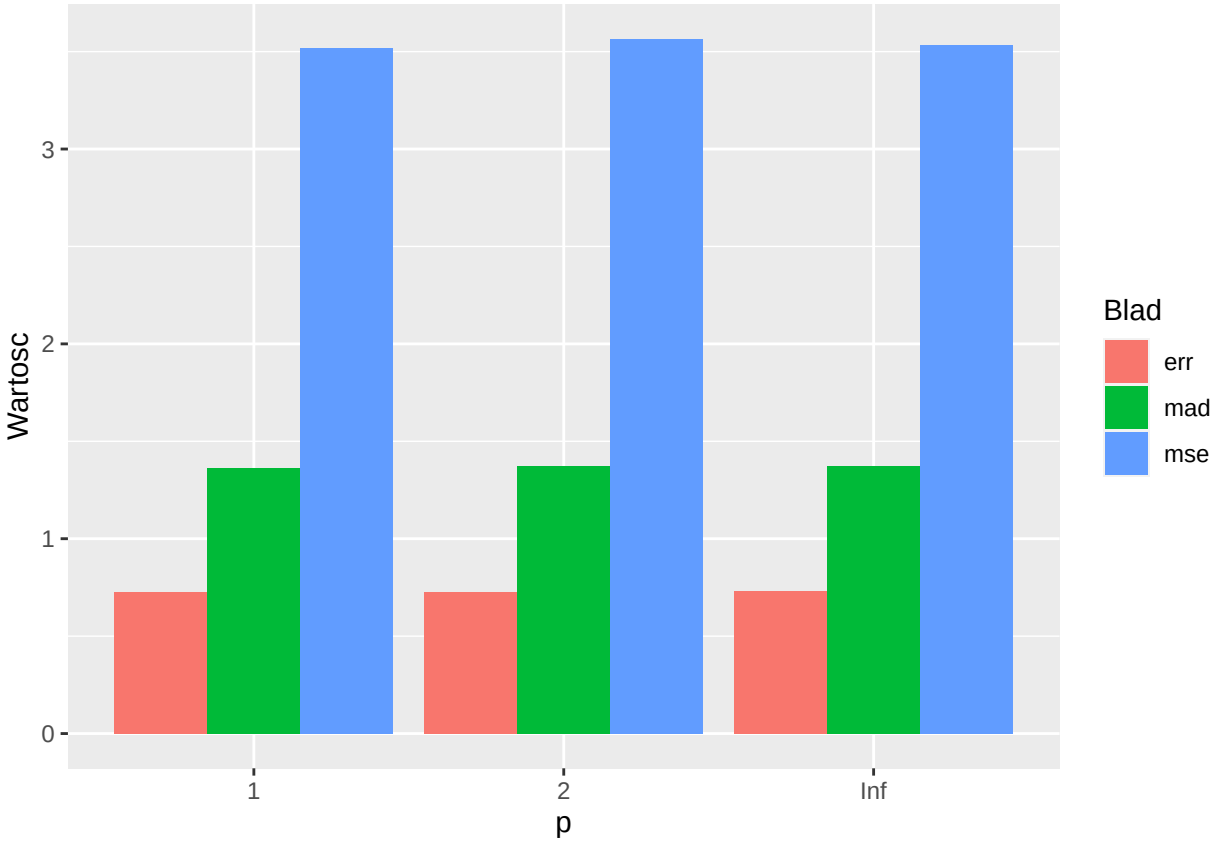
Wpływ parametru p

Do generowania wykresów błędów względem wybranej metryki użyjemy poniższej funkcji:

```
plotPErrs <- function(dataSet) {
  dataSet$p = as.factor(dataSet$p)

  grouped <- dataSet %>%
    group_by(p) %>%
    summarise(err=mean(err/100),
              mad=mean(mad),
              mse=mean(mse))

  grLong <- grouped %>%
    gather("Blad","Wartosc",-p)

  ggplot(grLong) + geom_col(aes(x=p, y=Wartosc, fill=Blad), position='dodge')
}
```

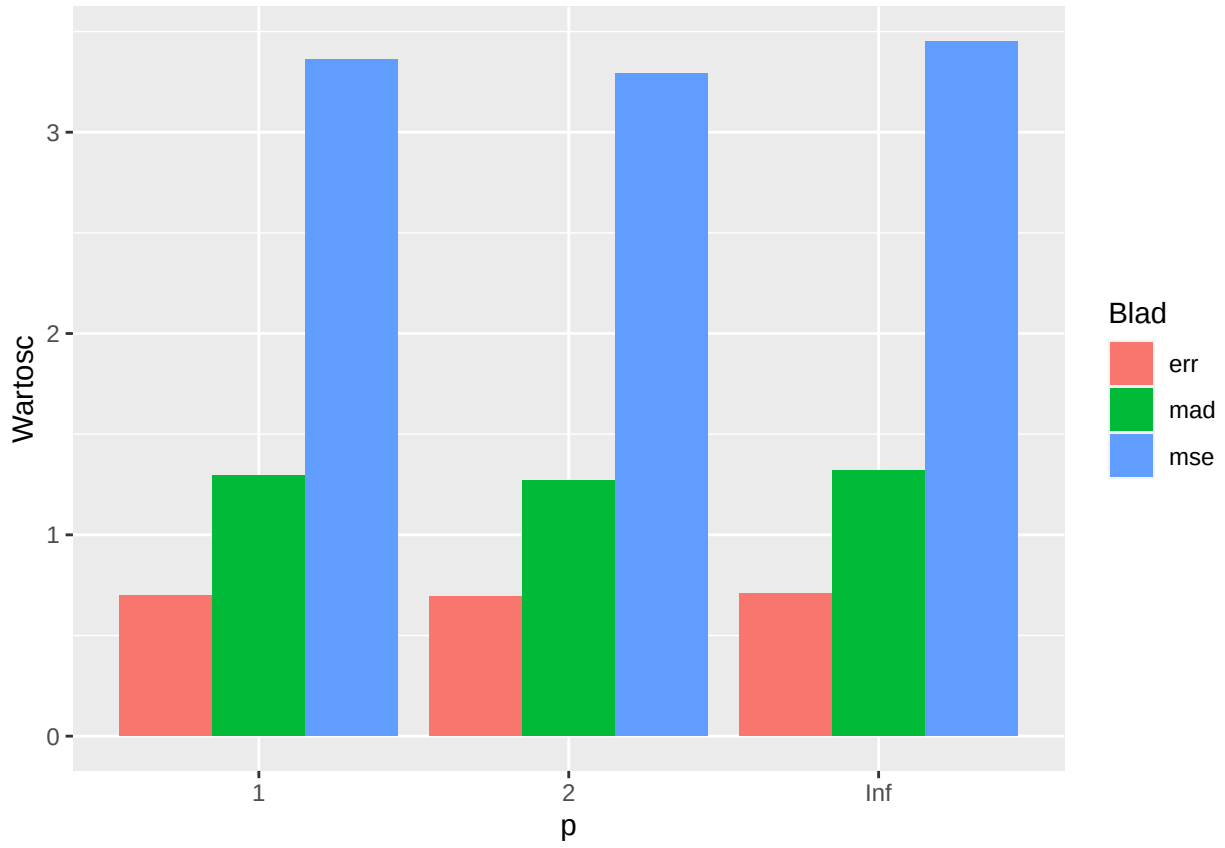
Dane `abalone.csv`

```
plotPErrs(abaloneKnnBenchmarks)
```



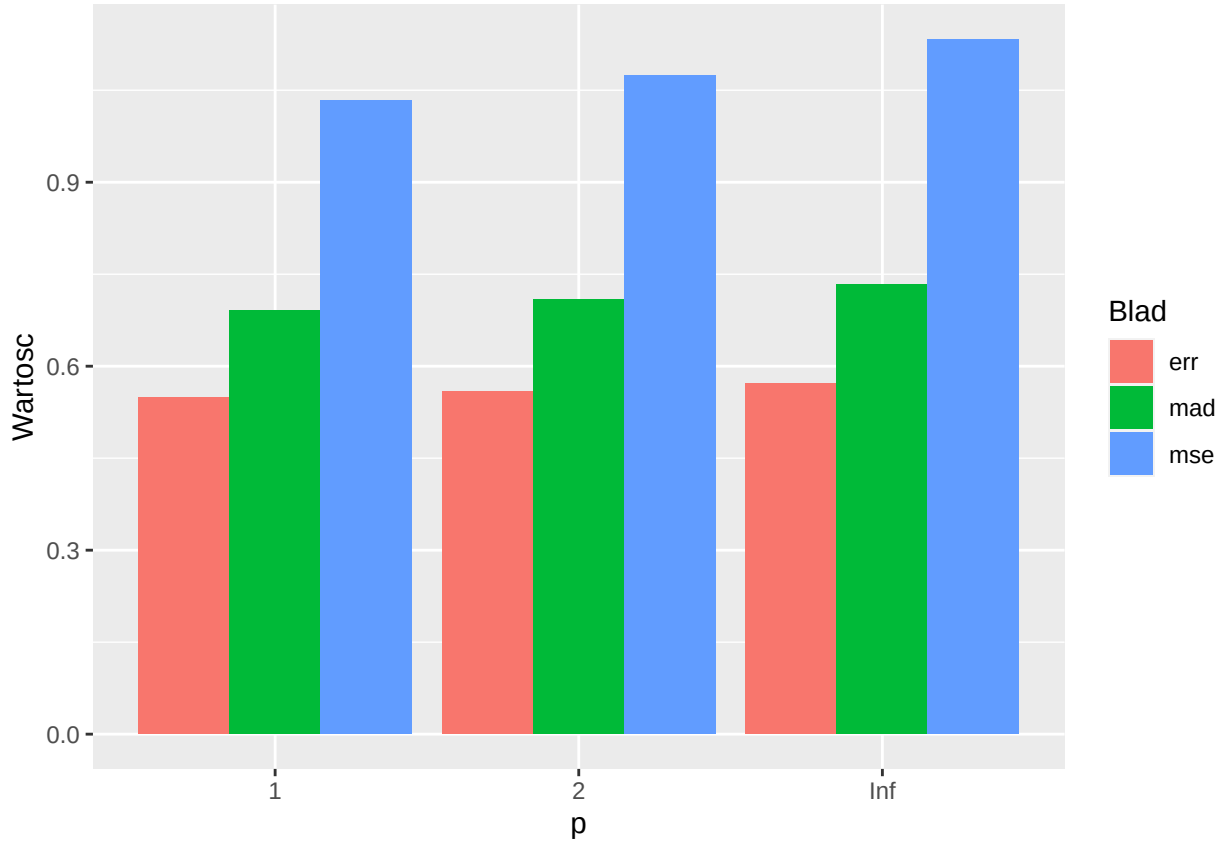
Dane `kinematics.csv`

```
plotPErrs(kinematicsKnnBenchmarks)
```



Dane `winequality-red.csv`

```
plotPErrs(redwineKnnBenchmarks)
```



Wnioski

Z wykresów wynika że użyta metryka nie wpływa istotnie na poprawność predykcji.

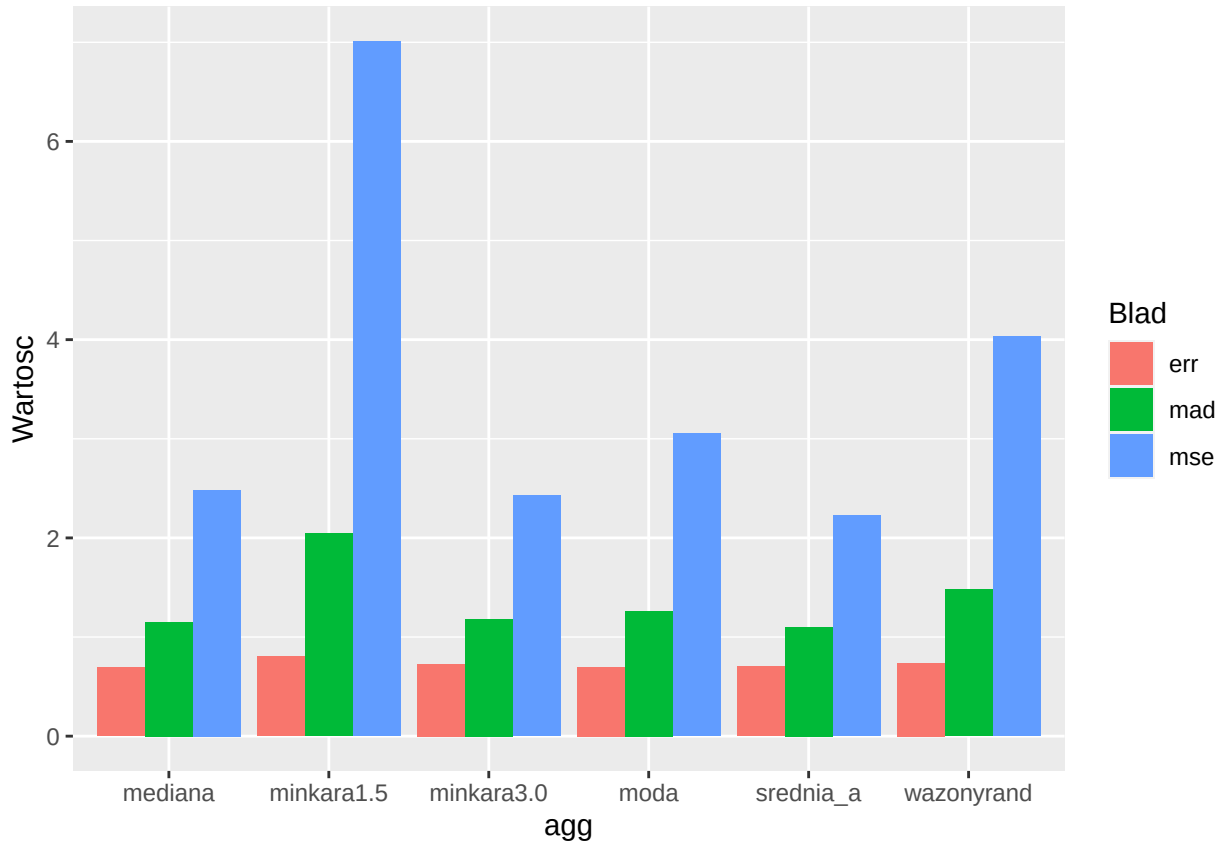
Wpływ agregatorów

Do generowania wykresów błędów względem wybranej funkcji agregującej użyjemy poniższej funkcji:

```
plotAggErrs <- function(dataSet) {
  grouped <- dataSet %>%
    mutate(agg = aggNames[aggId]) %>%
    group_by(agg) %>%
    summarise(err=mean(err/100),
              mad=mean(mad),
              mse=mean(mse))

  grLong <- grouped %>%
    gather("Blad","Wartosc",-agg)

  ggplot(grLong) + geom_col(aes(x=agg, y=Wartosc, fill=Blad), position='dodge')
}
```

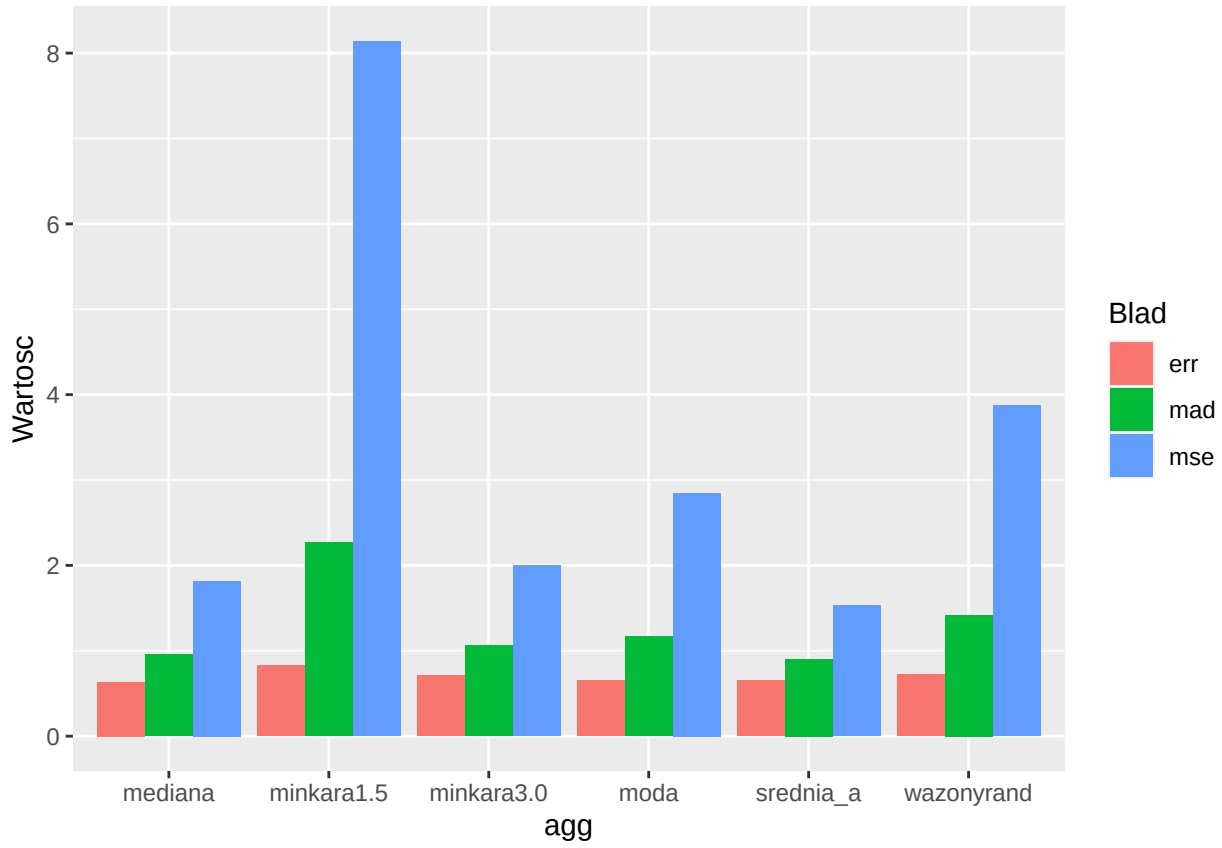
Dane `abalone.csv`

```
plotAggErrs(abaloneKnnBenchmarks)
```



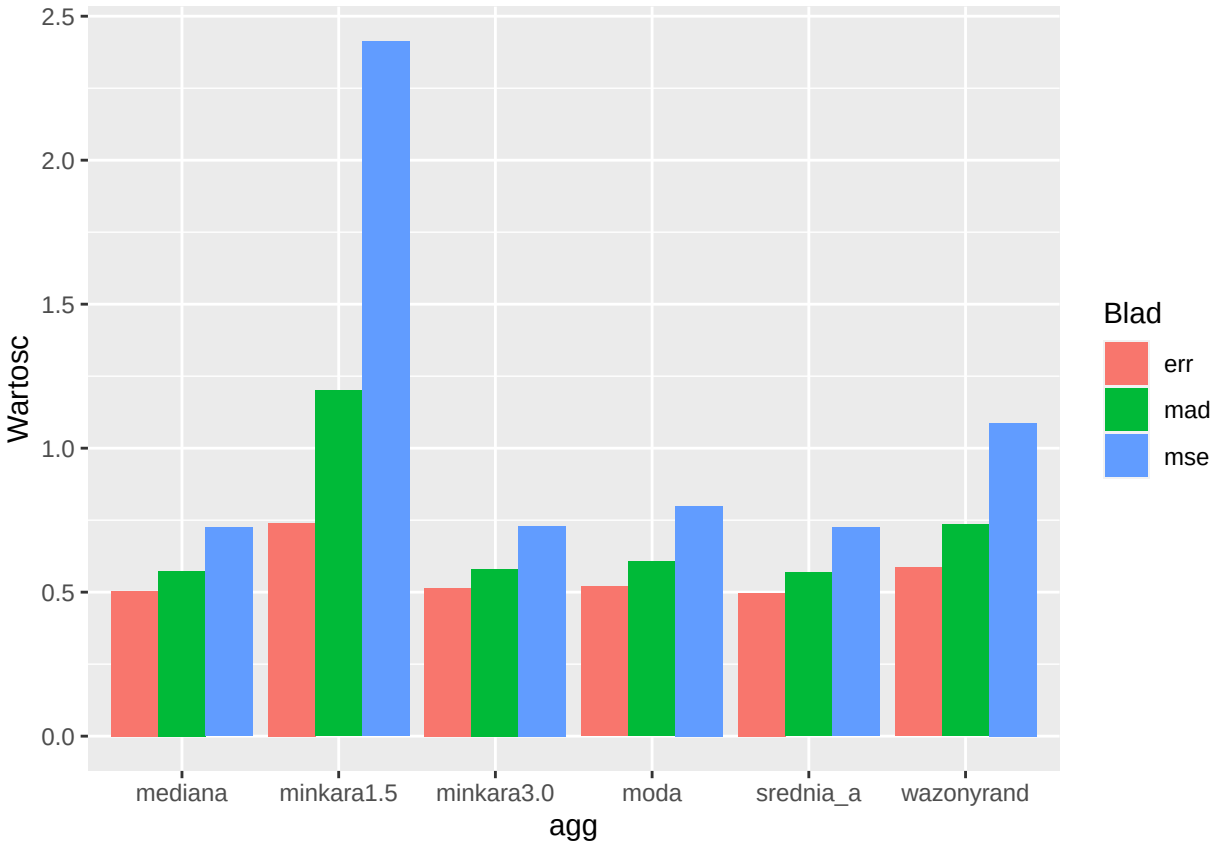
Dane `kinematics.csv`

```
plotAggErrs(kinematicsKnnBenchmarks)
```



Dane `winequality-red.csv`

```
plotAggErrs(redwineKnnBenchmarks)
```



Wnioski

Na wykresach bardzo odstaje agregacje `minkara1.5`, lecz ciężko mi uzasadnić co może być tego przyczyną. Zaproponowana przeze mnie agregacja `wazonyrand`, będąca losowym wyborem z silną tendencją bliższych środka wartości, wypada dosyć słabo, ale nadal znacznie lepiej od `minkara1.5`.

Porównanie `knn` z `randomForset::randomForest()`

W tej sekcji porównamy wskaźniki klasyfikacji dla zagregowanych benchmarków funkcji `knn` oraz pięciokrotnej kroswalidacji dla lasów losowych. Rozważymy te same zbiory danych co powyżej. Użyjemy do tego poniższych funkcji

```
forestCrossvalidation <- function(dataSet) {
  n <- nrow(dataSet)
  sampledData <- dataSet[sample(n, size = n, replace = FALSE),]
  err <- 0
  mad <- 0
  mse <- 0
  labels <- sampledData[,1]
  variables <- data.matrix(sampledData[, -1])

  for (i in 0:4) {
    start <- floor(i / 5 * n) + 1
    end <- floor((i + 1) / 5 * n)
    len <- end - start
```

```
    trainingSet <- sampledData[-end:-start,]
    testSet <- variables[start:end,]
    correctLabels <- labels[start:end]

    forest <- randomForest(response ~ ., data = trainingSet)
    resultLabels <- round(predict(forest, testSet))

    err <- err + sum(resultLabels != correctLabels) / len / 5
    mad <- mad + sum(abs(resultLabels - correctLabels)) / len / 5
    mse <- mse + sum((resultLabels - correctLabels) ^ 2) / len / 5
  }

  print("Wyniki dla klasyfikacji dla lasu losowego")
  print(paste("Procent blednej klasyfikacji: ", round((err * 100), 3), "%", sep = ""))
  print(paste("Blad bezwzgledny: ", round(mad, 4), sep = ""))
  print(paste("Blad srendiokwadratowy: ", round(mse, 4), sep = ""))
}

benchmarkedKnnCrossvalidation <- function(bDataSet) {
  errors <- sapply(bDataSet[,c("err", "mad", "mse")], mean)

  print("Wyniki dla klasyfikacji dla zagregowanego benchmarku knn")
  print(paste("Procent blednej klasyfikacji: ", round(errors[1], 3), "%", sep = ""))
  print(paste("Blad bezwzgledny: ", round(errors[2], 4), sep = ""))
  print(paste("Blad srendiokwadratowy: ", round(errors[3], 4), sep = ""))
}
```

Dane `abalone.csv`

```
forestCrossvalidation(abalone)
```

```
## [1] "Wyniki dla klasyfikacji dla lasu losowego"
## [1] "Procent blednej klasyfikacji: 70.686%"
## [1] "Blad bezwzgledny: 1.0501"
## [1] "Blad srendiokwadratowy: 1.935"
```

```
benchmarkedKnnCrossvalidation(abaloneKnnBenchmarks)
```

```
## [1] "Wyniki dla klasyfikacji dla zagregowanego benchmarku knn"
## [1] "Procent blednej klasyfikacji: 72.753%"
## [1] "Blad bezwzgledny: 1.3701"
## [1] "Blad srendiokwadratowy: 3.5394"
```

Dane `kinematics.csv`

```
forestCrossvalidation(kinematics)
```

```
## [1] "Wyniki dla klasyfikacji dla lasu losowego"
## [1] "Procent blednej klasyfikacji: 75.84%"
## [1] "Blad bezwzgledny: 1.069"
## [1] "Blad srendiokwadratowy: 1.8033"
```

```
benchmarkedKnnCrossvalidation(kinematicsKnnBenchmarks)
```

```
## [1] "Wyniki dla klasyfikacji dla zagregowanego benchmarku knn"
```

```
## [1] "Procent blednej klasyfikacji: 70.038%"
## [1] "Blad bezwzgledny: 1.2957"
## [1] "Blad srendiokwadratowy: 3.3681"
```

Dane `winequality-red.csv`

```
forestCrossvalidation(redwine)
```

```
## [1] "Wyniki dla klasyfikacji dla lasu losowego"
## [1] "Procent blednej klasyfikacji: 38.994%"
## [1] "Blad bezwzgledny: 0.418"
## [1] "Blad srendiokwadratowy: 0.4756"
```

```
benchmarkedKnnCrossvalidation(redwineKnnBenchmarks)
```

```
## [1] "Wyniki dla klasyfikacji dla zagregowanego benchmarku knn"
## [1] "Procent blednej klasyfikacji: 55.985%"
## [1] "Blad bezwzgledny: 0.7107"
## [1] "Blad srendiokwadratowy: 1.0798"
```

Podsumowanie porównania

Z powyższych wskaźników wynika że funkcja lasów losowych `randomForset::randomForest()` na ogół skuteczniej przewiduje niż `knn` zagregowane po każdym możliweym parametrze. Lecz mimo to, w jednym ze zbiorów danych to właśnie `knn` wypadło delikatnie lepiej. Warto zauważyć że przy agregacji benchmarków `knn` nie zostały wykluczone mocno negatywnie wpływające na wyniki parametry takie jak `k=1` lub funkcja agregująca `minkara1.5`. Prawdopodobnie wykluczając je `knn` wypadłoby lepiej na tle lasów losowych.