Statystyczna analiza danych - projekt zaliczeniowy 2

Marta Solarz

1. Eksploracja

Zmienne objaśniające:

X_train.csv i X_test.csv - wiersze to poszczególne komórki, kolumny to geny, a wartości to poziom ekspresji.

Zmienna objaśniana:

y_train.csv - ilość pewnego typu białka powierzchniowego w poszczególnych komórkach.

(a)

Sprawdź, ile obserwacji i zmiennych zawierają wczytane dane treningowe oraz testowe. Przyjrzyj się typom zmiennych i, jeśli uznasz to za słuszne, dokonaj odpowiedniej konwersji przed dalszą analizą. Upewnij się, czy dane są kompletne.

```
x_train <- read.csv('D:/SAD/X_train.csv')
dim(x_train)
```

```
## [1] 3794 9000
```

```
# A zatem 3794 wierszy i 9000 kolumn.

x_test <- read.csv('D:/SAD/X_test.csv')
dim(x_test)
```

```
## [1]  670 9000
```

```
# A zatem 670 wierszy i 9000 kolumn.

y_train <- read.csv('D:/SAD/y_train.csv')
dim(y_train)
```

```
## [1] 3794    2
```

```
# A zatem 3794 wierszy i 2 kolumny.

sum <- nrow(x_train) + nrow(x_test)
nrow(x_train)/sum
```

```
## [1] 0.8499104
```

Możemy zauważyć, że wymiary danych się zgadzają - liczba zmiennych objaśniających (liczba kolumn x_train i x_test) jest taka sama. Do zbioru treningowego trafiło 3794 obserwacji, a do zbioru testowego 670, a zatem jest to podział 85% do 15%. Liczba wierszy (obserwacji) w y_train jest poprawna, ponieważ jest równa liczbie obserwacji w x_train.

```
# Typy zmiennych
# str(x_train)
x_train[!sapply(x_train, is.numeric)]
```

```
## ramka danych z zerową liczbą kolumn oraz 3794 wierszami
```

```
# A zatem wszystkie kolumny są numeryczne - to wydaje się ok,
# jeśli mówimy o wartości ekspresji genu.
# summary(x_train)

# str(x_test)
x_test[!sapply(x_test, is.numeric)]
```

```
## ramka danych z zerową liczbą kolumn oraz 670 wierszami
```

```
# A zatem wszystkie kolumny są numeryczne (zgadza się to z x_train).

str(y_train)
```

```
## 'data.frame':    3794 obs. of  2 variables:
##  $ Id      : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Expected: num  3.091 0.89 0.529 1.448 6.213 ...
```

```
# Id nie jest liczbą, tylko znakiem porządkowym
# (rozróżnieniem jakościowym obserwacji), a zatem zamieńmy na typ jakościowy.
y_train$Id <- as.character(y_train$Id)
```

(b)

Zbadaj rozkład empiryczny zmiennej objaśnianej (przedstaw kilka podstawowych statystyk, do analizy dołącz histogram lub wykres estymatora gęstości).

```
# Podstawowe statystyki
summary(y_train$Expected)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.8081  1.1566  2.3327  2.4231  7.4834
```

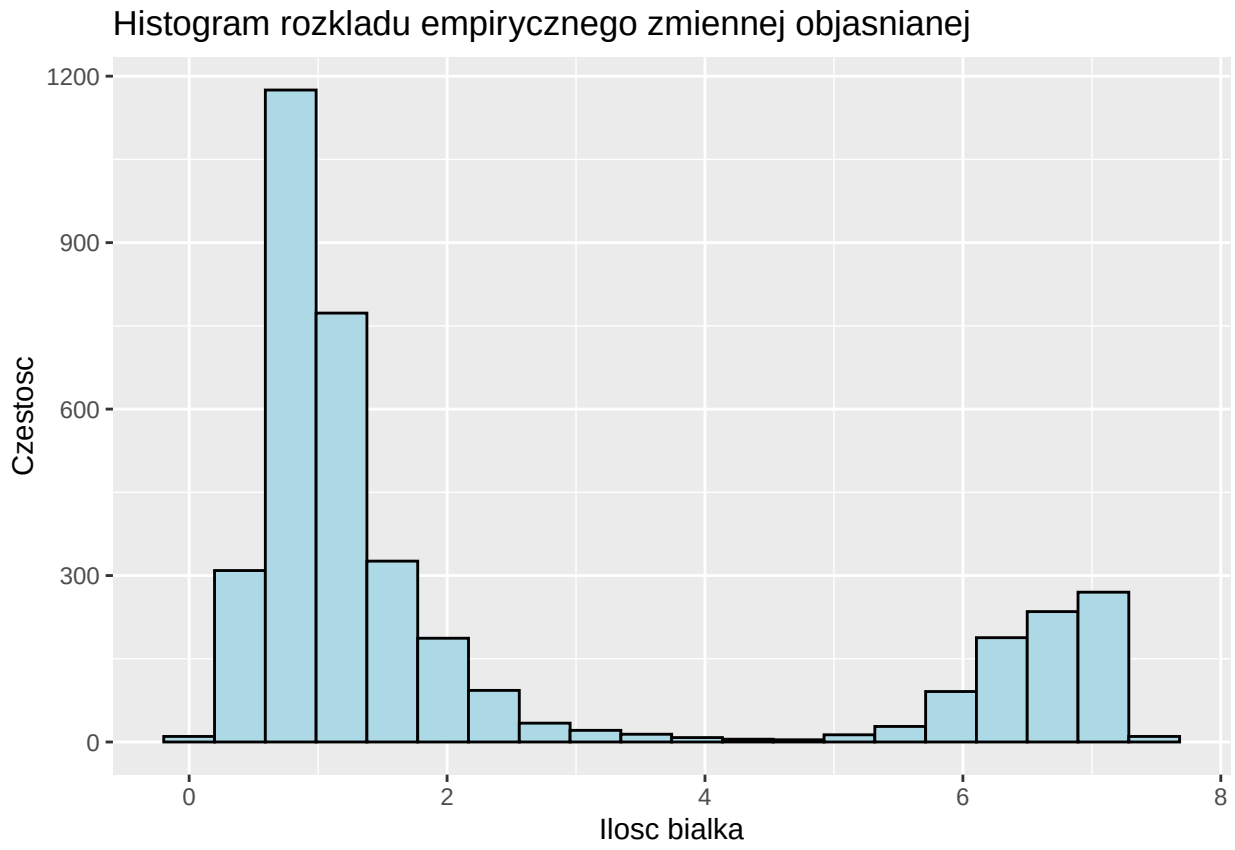
```
var(y_train$Expected)
```

```
## [1] 5.443485
```

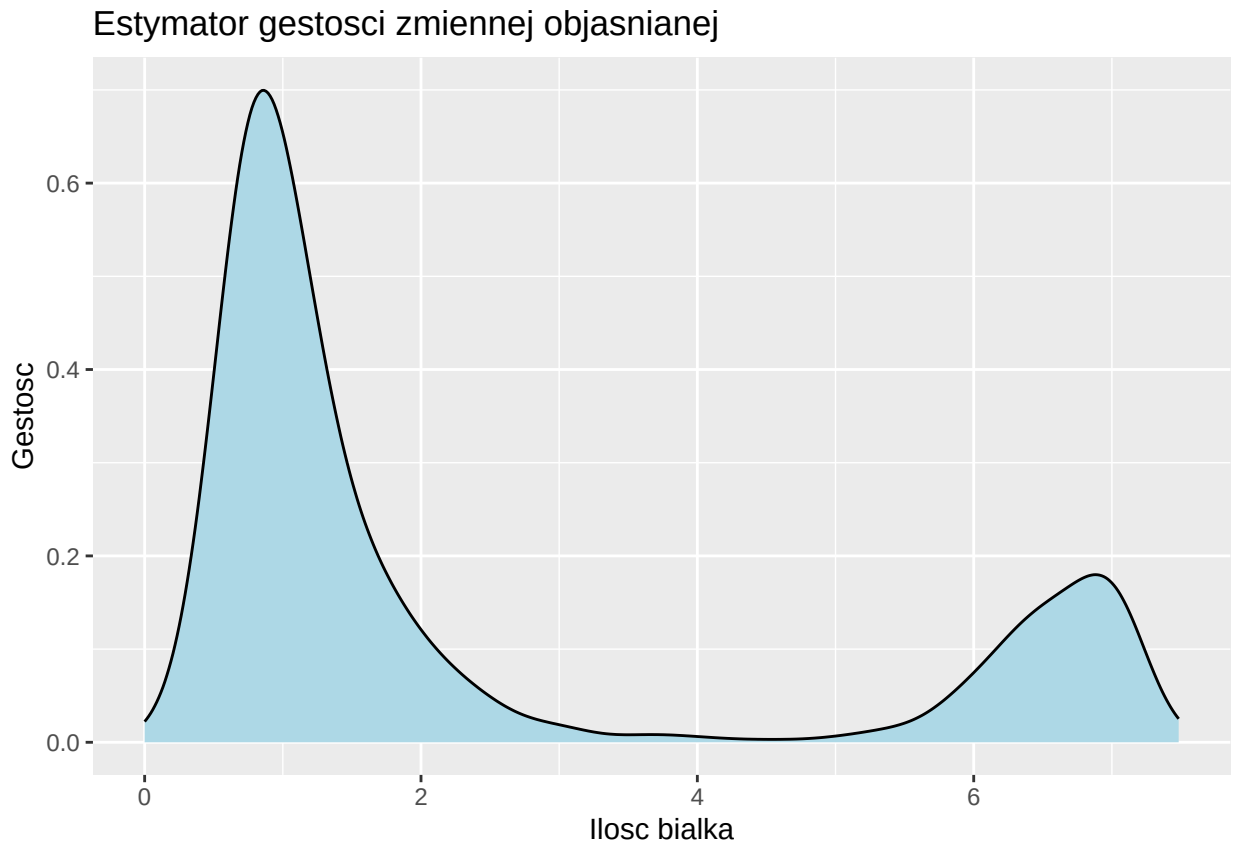
```r
sd(y_train$Expected)
```

```
## [1] 2.333128
```

```r
# Histogram
library(ggplot2)

ggplot(y_train, aes(x = Expected)) +
  geom_histogram(fill = "lightblue", color = "black", bins = 20) +
  labs(title = "Histogram rozkladu empirycznego zmiennej objasnianej",
       x = "Ilosc bialka", y = "Czestosc")
```



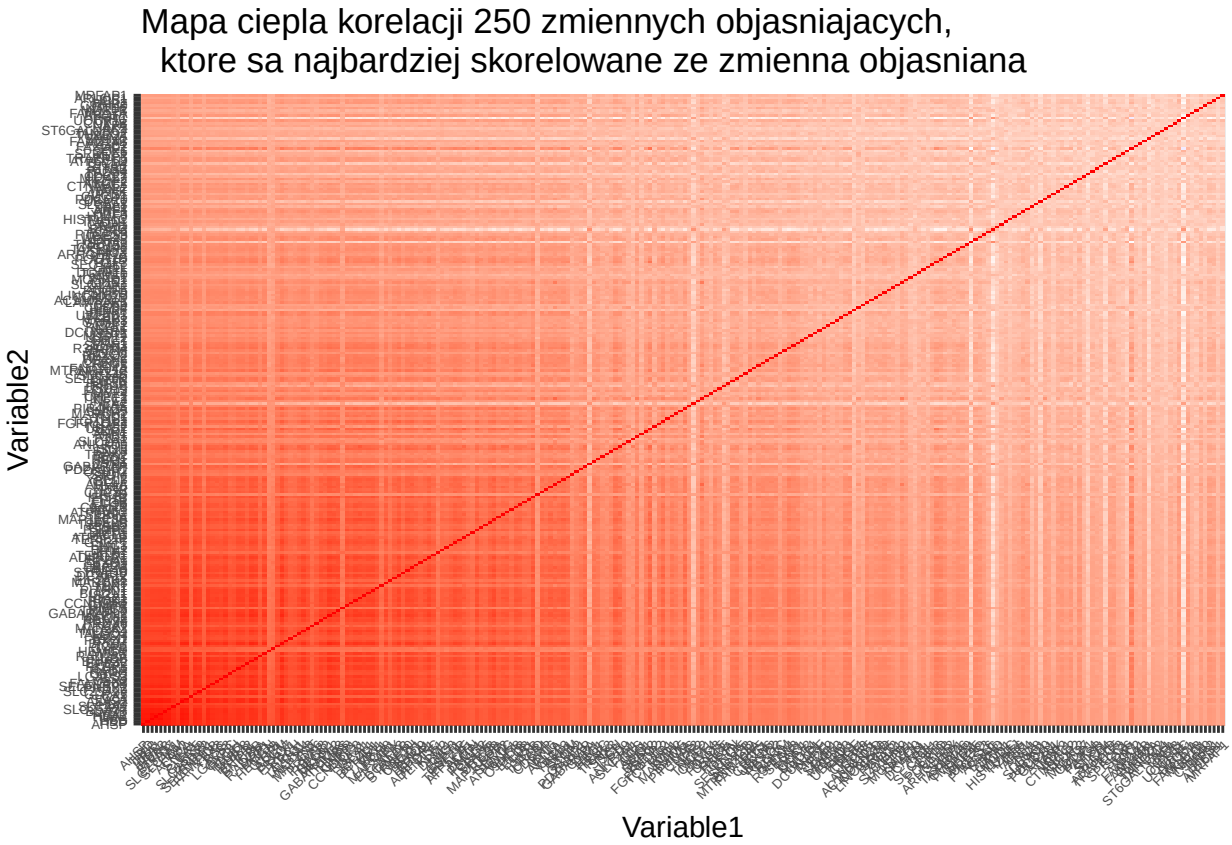

```r
# Wykres estymatora gęstości
ggplot(y_train, aes(x = Expected)) +
  geom_density(fill = "lightblue", color = "black") +
  labs(title = "Estymator gestosci zmiennej objasnianej",
       x = "Ilosc bialka", y = "Gestosc")
```



(c)

Wybierz 250 zmiennych objaśniających najbardziej skorelowanych ze zmienną objaśnianą. Policz korelację dla każdej z par tych zmiennych. Zilustruj wynik za pomocą mapy ciepła.

```
# Policzenie korelacji wszystkich zmiennych kolejno ze zmienną objaśnianą
# i wybranie 250 najbardziej skorelowanych:
library(reshape2)
cor_matrix <- cor(x_train, y_train$Expected)
cor_values <- cor_matrix[,1]
top_cor_vars <- names(sort(cor_values, decreasing = TRUE))[1:250]

cor_pairs <- cor(x_train[, top_cor_vars])
cor_df <- reshape2::melt(cor_pairs)
colnames(cor_df) <- c("Variable1", "Variable2", "Value")

ggplot(data = cor_df, aes(x = Variable1, y = Variable2, fill = Value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red") +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1, size = 5),
        axis.text.y = element_text(size = 5),
        panel.grid = element_blank(),
        legend.position = "none") +
  labs(title = "Mapa ciepla korelacji 250 zmiennych objasniajacych,
  ktore są najbardziej skorelowane ze zmienna objasniana")
```



2. ElasticNet

(a)

Wyszukaj i przedstaw w raporcie informacje o modelu ElasticNet. Opisz parametry, które są w nim estymowane, optymalizowaną funkcję oraz hiperparametry, od których ona zależy. Dla jakich wartości hiperparametrów otrzymujemy regresję grzbietową, a dla jakich lasso?

ElasticNet jest modelem łączącym regresję grzbietową i lasso. Wykorzystuje kombinację tych dwóch metod, umożliwiając jednoczesne przeprowadzenie selekcji cech (Lasso) oraz redukcji wielowymiarowości (grzbietowa regresja). Poprawia ona ograniczenia lasso (jeśli dane są silnie skorelowane, to lasso ma tendencję do wybierania jednej zmiennej z takich grup i ignorowania pozostałych) poprzez dodanie w karze wyrażenia kwadratowego $||\beta||^2$, które stosowane jest w regresji grzbietowej.

Funkcja optymalizowana ma postać: $\hat{\beta} = argmin_\beta(||y - X\beta||^2 + \lambda_2||\beta||^2 + \lambda_1||\beta||_1)$

Parametry estymowane w modelu ElasticNet obejmują współczynniki regresji dla każdej cechy oraz parametr regularyzacji. Funkcja straty jest sumą dwóch składników: błędu kwadratowego między przewidywanymi wartościami a rzeczywistymi wartościami zmiennych zależnych i kombinacji L_1 i L_2 norm współczynników regresji.

Hiperparametry definiowane w R, od których zależy funkcja straty w modelu ElasticNet, to:

- Alpha: Kontroluje proporcję między regularyzacją L_1 (lasso) a regularyzacją L_2 (grzbietowa regresja). Wartości alpha w zakresie od 0 do 1 pozwalają na różne proporcje kombinacji tych dwóch metod.

- Lambda: Parametr regularyzacji, który kontroluje siłę regularyzacji. Wyższe wartości lambdy prowadzą do silniejszej regularyzacji, co może prowadzić do redukcji współczynników regresji do zera. Niższe wartości lambdy pozwalają na mniejszą regularyzację, co może prowadzić do modelu bardziej dopasowanego do danych treningowych.

Dla wartości hiperparametru alpha równego 0 otrzymujemy regresję grzbietową, która ma tendencję do pomniejszania wpływu mniej istotnych cech, ale nie redukuje ich do zera. Regresja grzbietowa zachowuje wszystkie cechy w modelu, chociaż ich wpływ jest zmniejszony przez regularyzację L_2.

Dla wartości hiperparametru alpha równego 1 otrzymujemy regresję lasso, która ma zdolność do selekcji cech poprzez redukcję współczynników regresji do zera. Lasso może wyeliminować nieistotne cechy, co prowadzi do prostszego modelu zawierającego tylko istotne zmienne.

Elastyczna kombinacja obu metod w ElasticNet pozwala na wykorzystanie ich zalet jednocześnie, umożliwiając selekcję cech i redukcję wielowymiarowości w jednym modelu.

(b)

Zdefiniuj siatkę (grid) hiperparametrów, opartą na co najmniej trzech wartościach każdego z hiperparametrów. Zadbaj o to, by w siatce znalazły się konfiguracje hiperparametrów odpowiadające regresji grzbietowej i lasso. Użyj walidacji krzyżowej do wybrania odpowiednich hiperparametrów (o liczbie podzbiorów użytych w walidacji krzyżowej należy zdecydować samodzielnie oraz uzasadnić swój wybór).

```
# Tworzenie podzbiorów walidacji krzyżowej
# (aby były takie same dla ElasticNet i potem RandomForest)

library(caret)
```

```
## Ładowanie wymaganego pakietu: lattice
```

```
trainData <- cbind(x_train, y_train$Expected)

set.seed(123)
control <- trainControl(method = "cv", number = 10)
```

Wybrałam 10 podzbiorów walidacji krzyżowej, ponieważ:

- ta liczba stanowi kompromis pomiędzy czasem obliczeń a precyzją - im więcej jest podzbiorów, tym większa precyzja, ale także dłuższy czas obliczeń;
- jest to standardowa liczba podzbiorów wybierana do walidacji.

```
library(glmnet)
```

```
## Ładowanie wymaganego pakietu: Matrix
```

```
## Loaded glmnet 4.1-7
```

```
alpha_values <- c(0, 0.5, 1) # 0 - regresja grzbietowa, 1 - lasso
lambda_values <- c(0.1, 0.01, 0.001)

hyperparameters_elasticNet <- expand.grid(alpha = alpha_values,
                                          lambda = lambda_values)

start <- Sys.time()

set.seed(123)
model_elasticNet <- train(`y_train$Expected` ~ ., data = trainData,
               method = "glmnet", trControl = control,
               tuneGrid = hyperparameters_elasticNet)

stop <- Sys.time()

print(stop-start)
```

```
## Time difference of 2.25537 mins
```

```
# Najlepsze parametry:
model_elasticNet$bestTune
```

```
##   alpha lambda
## 5   0.5   0.01
```

(c)

Podaj błąd treningowy i walidacyjny modelu (należy uśrednić wynik względem wszystkich podzbiorów wyróżnionych w walidacji krzyżowej).

```
# Błąd treningowy
train_predictions_elasticNet <- predict(model_elasticNet, newdata = x_train)
train_error_elasticNet <- sqrt(mean((train_predictions_elasticNet - y_train$Expected)^2))

# Błąd walidacyjny
validation_error_elasticNet <- mean(model_elasticNet$resample$RMSE)

print(paste("Błąd treningowy (RMSE) dla elasticNet:",
            train_error_elasticNet))
```

```
## [1] "Błąd treningowy (RMSE) dla elasticNet: 0.449306901546785"
```

```
print(paste("Błąd walidacyjny (RMSE) dla elasticNet:",
            validation_error_elasticNet))
```

```
## [1] "Błąd walidacyjny (RMSE) dla elasticNet: 0.517753345602256"
```

3. Lasy losowe

(a)

Spośród wielu hiperparametrów charakteryzujących model lasów losowych wybierz trzy różne. Zdefiniuj trójwymiarową siatkę przeszukiwanych kombinacji hiperparametrów i za pomocą walidacji krzyżowej wybierz ich optymalne (w kontekście wykonywanej predykcji) wartości. Wykorzystany przy walidacji krzyżowej podział danych powinien być taki sam, jak w przypadku ElasticNet.

```
library(ranger)

mtry_values <- c(2, 4, 6)
splitrule_values <- c("variance", "extratrees", "maxstat")
min_node_size_values <- c(3, 7, 10)

hyperparameters_rf <- expand.grid(.mtry = mtry_values,
                                  .splitrule = splitrule_values,
                                  .min.node.size = min_node_size_values)

start <- Sys.time()

set.seed(123)
model_rf <- train(`y_train$Expected` ~ ., data = trainData, num.trees = 100,
                  method = "ranger", trControl = control,
                  tuneGrid = hyperparameters_rf, importance = "impurity")

stop <- Sys.time()

print(stop-start)
```

```
## Time difference of 39.09744 mins
```

```
# Najlepsze parametry:
model_rf$bestTune
```

```
##    mtry splitrule min.node.size
## 19    6  variance             3
```

```
# Błąd treningowy
train_predictions_rf <- predict(model_rf, newdata = x_train)
train_error_rf <- sqrt(mean((train_predictions_rf - y_train$Expected)^2))

# Błąd walidacyjny
validation_error_rf <- mean(model_rf$resample$RMSE)

print(paste("Błąd treningowy (RMSE) dla lasów losowych:", train_error_rf))
```

```
## [1] "Błąd treningowy (RMSE) dla lasów losowych: 0.282728237847841"
```

```
print(paste("Błąd walidacyjny (RMSE) dla lasów losowych:", validation_error_rf))
```

```
## [1] "Błąd walidacyjny (RMSE) dla lasów losowych: 0.600009532446207"
```

(b)

Zrób podsumowanie tabelaryczne wyników, jakie otrzymywały metody w walidacji krzyżowej w obu rozważanych modelach. (Porównanie to jest powodem, dla którego zależy nam na zastosowaniu tych samych podziałów). Określ, który model wydaje Ci się najlepszy (uzasadnij swój wybór). Do porównania dołącz podstawowy model referencyjny, który dowolnym wartościom zmiennych objaśniających przypisuje średnią arytmetyczną zmiennej objaśnianej.

```
# Model referencyjny
mean_reference_model <- function(data, target_variable) {
  mean_target <- mean(target_variable)
  predictions <- rep(mean_target, nrow(data))
  return(predictions)
}

reference_model_prediction <- mean_reference_model(
  trainData,
  trainData$`y_train$Expected`
)

reference_model_error <- sqrt(
  mean((reference_model_prediction - y_train$Expected)^2)
)

reference_model_error
```

```
## [1] 2.33282
```

```
model_elasticNet$resample$RMSE
```

```
##  [1] 0.5298169 0.5961894 0.5192765 0.5221780 0.4778045 0.4687631 0.5354846
##  [8] 0.5119645 0.5114623 0.5045937
```

```
model_rf$resample$RMSE
```

```
##  [1] 0.5829410 0.5674257 0.6352647 0.5896619 0.6247279 0.6213011 0.5853723
##  [8] 0.6129841 0.6128382 0.5675785
```

Błąd RMSE w modelu referencyjnym wynosi: 2.33282.

	RMSE ElasticNet	RMSE RandomForest
1	0.5298169	0.5829410
2	0.5961894	0.5674257
3	0.5192765	0.6352647
4	0.5221780	0.5896619
5	0.4778045	0.6247279
6	0.4687631	0.6213011
7	0.5354846	0.5853723
8	0.5119645	0.6129841
9	0.5114623	0.6128382

	RMSE ElasticNet	RMSE RandomForest
10	0.5045937	0.5675785

Na podstawie wyników dla 10 podzbiorów walidacji krzyżowej można uznać, że lepszy okazał się model ElasticNet (w 9/10 przypadkach RMSE był niższy dla ElasticNet). Biorąc pod uwagę fakt, że dane zawierały 9000 zmiennych objaśniających i nie całe 4000 obserwacji, model ElasticNet może być korzystniejszy niż model RandomForest ze względu na swoje właściwości.

ElasticNet jest techniką regularizacji, która łączy w sobie regresję Lasso i regresję grzbietową. Lasso promuje rzadkie modele, co oznacza, że może przypisać wartości zerowe wielu zmiennym objaśniającym, eliminując te, które nie wniosły znaczącego wkładu do modelu. Z drugiej strony, regresja grzbietowa redukuje wielkość współczynników, ale nie eliminuje ich całkowicie.

Dlatego w przypadku, gdy dane zawierają dużą liczbę zmiennych w stosunku do liczby obserwacji, model ElasticNet może lepiej radzić sobie z wyborem odpowiednich zmiennych objaśniających (w konsekwencji zmieniejszając ryzyko overfittingu). Poprzez promowanie rzadkich modeli, ElasticNet może efektywnie selekcjonować istotne zmienne, ograniczając wpływ szumowych lub nieistotnych zmiennych na predykcję.

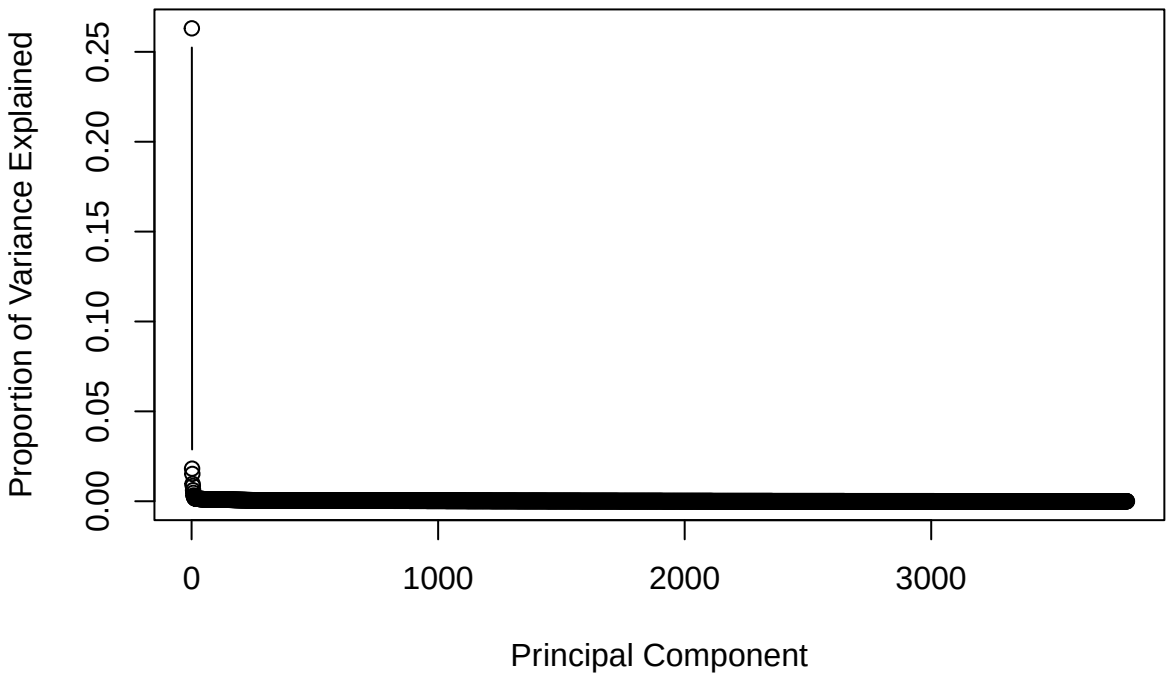
Model RandomForest może być bardziej skłonny do dopasowania się do danych, co może prowadzić do nadmiernego dopasowania w przypadku niewystarczającej liczby obserwacji. Ponadto, w przypadku dużego zbioru zmiennych, wytrenowanie modelu random forest może być bardziej złożone obliczeniowo, co widoczne było w tym projekcie.

4. Predykcja na zbiorze testowym

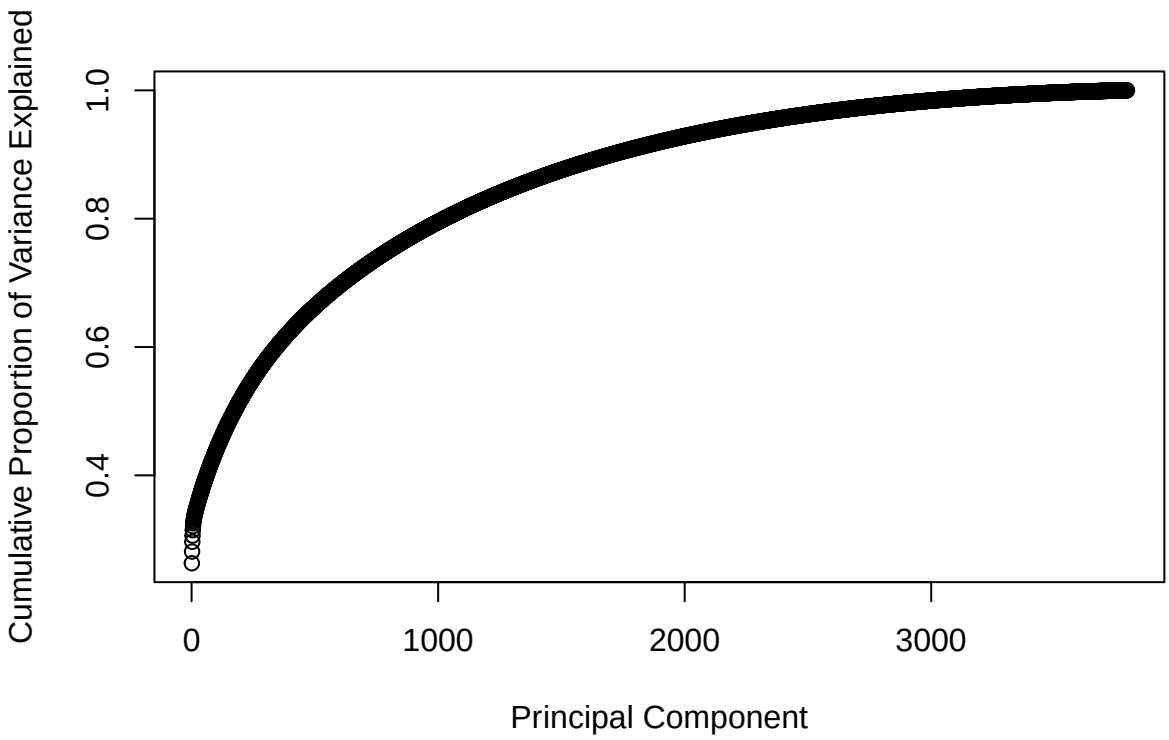
```
# Transformacja PCA danych
pca <- prcomp(x_train, retx=T)

importance <- pca$sdev^2 / sum(pca$sdev^2)
importance[1:30]
```

```
##  [1] 0.263063566 0.018128959 0.015181631 0.009694783 0.008499649 0.006135126
##  [7] 0.004651354 0.003199594 0.002853449 0.002365191 0.002128318 0.001819471
## [13] 0.001680479 0.001589017 0.001581779 0.001546975 0.001495590 0.001454752
## [19] 0.001444994 0.001419232 0.001404355 0.001386528 0.001372819 0.001365640
## [25] 0.001355411 0.001340341 0.001332144 0.001318719 0.001313459 0.001303959
```

```
plot(importance, xlab = "Principal Component",
             ylab = "Proportion of Variance Explained",
             type = "b")
```




```
plot(cumsum(importance), xlab = "Principal Component",
              ylab = "Cumulative Proportion of Variance Explained",
              type = "b")
```




```
train.data_x <- data.frame(pca$x[,1:2900])
ncol(train.data_x)
```

```
## [1] 2900
```

```
train.data_y = y_train$Expected

test.data_x <- data.frame(predict(pca, newdata = x_test))
test.data_x <- test.data_x[,1:2900]
ncol(test.data_x)
```

```
## [1] 2900
```

Wybrałam 2900 pierwszych składowych głównych, aby uniknąć szumu znajdującego się w dalszych składowych. Dla takiej liczby składowych uzyskałam najwyższy wynik predykcji.

Do predykcji y_test wykorzystam model LightGBM oparty na technologii „gradient boosting", która polega na sekwencyjnym trenowaniu wielu słabych modeli (drzew decyzyjnych) w celu poprawy predykcji. Główną ideą tego podejścia jest wykorzystanie gradientów funkcji straty do modyfikowania kolejnych modeli w taki sposób, aby poprawiały one niedokładności poprzednich modeli.

Wyróżniającymi się cechami tego modelu są m.in:

- wykorzystanie GOSS, który jest techniką próbkowania opartą na gradientach. GOSS wybiera instancje o dużych gradientach (większe znaczenie) i próbkuje instancje o małych gradientach (mniejsze znaczenie). Dzięki temu osiąga się równowagę między ważnością instancji a wydajnością obliczeniową.

- stosowanie EFB, który polega na grupowaniu unikalnych wartości kategorycznych cech w jedno. Redukuje to ilość kolumn danych i przyspiesza trening modelu.
- używanie histogramów do przyspieszenia procesu treningu. Zamiast sortować dane dla każdego podziału, LightGBM tworzy histogramy dla każdej cechy i wybiera najlepszy podział na ich podstawie.
- używanie wzrostu drzewa wzdłuż liści, co oznacza, że drzewa są rozwijane warstwami, a nie poziomami. Ta technika może prowadzić do większej złożoności modelu, ale jednocześnie może zapewnić lepszą dokładność predykcji.

Dzięki tym cechom LightGBM jest w stanie osiągnąć wysokie wyniki RMSE dla naszego zbioru danych.

```
library(lightgbm)
```

```
## Ładowanie wymaganego pakietu: R6
```

```
train_data <- lgb.Dataset(data = as.matrix(train.data_x),
                          label = train.data_y)

params <- list(
  objective = "regression",  # Funkcja celu dla regresji
  boosting_type = "gbdt",  # Typ modelu boosting
  metric = "rmse"  # Metryka oceny jakości modelu
)

start <- Sys.time()
model <- lgb.train(
  params = params,
  data = train_data,
  num_boost_round = 250,
  verbose = 1
)
```

```
## Warning in lgb.train(params = params, data = train_data, num_boost_round = 250,
## : lgb.train: Found the following passed through '...': num_boost_round. These
## will be used, but in future releases of lightgbm, this warning will become an
## error. Add these to 'params' instead. See ?lgb.train for documentation on how
## to call this function.

## [LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.187251 se
## You can set `force_col_wise=true` to remove the overhead.
## [LightGBM] [Info] Total Bins 739500
## [LightGBM] [Info] Number of data points in the train set: 3794, number of used features: 2900
## [LightGBM] [Info] Start training from score 2.332684
```

```
stop <- Sys.time()
print(stop-start)
```

```
## Time difference of 29.13749 secs
```

```
predictions <- predict(model, as.matrix(test.data_x))

results <- data.frame(Id = seq(0,(nrow(x_test) - 1)), Expected = predictions)
write.csv(results, "D:/SAD/421826_predykcja.csv", row.names = FALSE)
```