Лабораторная работа 1 “Предобработка данных”

12 сентября 2025 г.

Цель работы

Исследование влияния различных методов обработки пропущенных значений на качество линейной регрессионной модели и сравнение эффективности методов предобработки данных.

Задачи

1. Реализовать функцию фильтрации данных методом Савицкого-Голея
2. Реализовать функцию нормализации данных методом IQR
3. Создать функцию для генерации пропущенных значений в датасете
4. Реализовать функцию восстановления пропусков двумя методами:
    - Заполнение средним значением по столбцу
    - Заполнение средним значением по скользящему окну
5. Провести сравнительный анализ качества моделей:
    - Модель, обученная на исходных данных без пропусков
    - Модель, обученная на данных с восстановлением средним
    - Модель, обученная на данных с восстановлением скользящим средним
6. Визуализировать результаты и сделать выводы

Исходные данные

- Датасет: House Prices или аналогичный регрессионный датасет
- Целевая переменная: `target`
- Факторы: `feature1, feature2, ..., featureN`
- Процент пропусков: 20% от общего числа наблюдений

Требования к отчету

- Описание датасетов и выбранных переменных
- Обоснование выбора методов предобработки
- Визуализация результатов на каждом этапе
- Сравнительный анализ метрик качества моделей
- Выводы и рекомендации по использованию методов

Импорт библиотек

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from scipy.signal import savgol_filter
import matplotlib.pyplot as plt
import seaborn as sns

# For displaying Russian characters in matplotlib
plt.rcParams['font.family'] = 'DejaVu Sans'
plt.rcParams['axes.unicode_minus'] = False
```

Листинг 1: Import necessary libraries (Python)

```r
# install.packages(c("dplyr", "ggplot2", "signal", "zoo", "caret", "tidyr"))

library(dplyr)
library(ggplot2)
library(signal)
library(zoo)
library(caret)
library(tidyr)

# For displaying Russian characters
Sys.setlocale("LC_CTYPE", "russian")
```

Листинг 2: Import necessary packages (R)

Функция фильтрации Савицкого-Голея

```python
def savitzky_golay_filter(data, window_size=11, poly_order=3):
    """
    Apply Savitzky-Golay filter to data

    Parameters:
    data: array-like - input data
    window_size: int - window size (must be odd)
    poly_order: int - polynomial order

    Returns:
    filtered_data: array - filtered data
    """
    if len(data) < window_size:
        window_size = len(data) if len(data) % 2 == 1 else len(data) - 1

    return savgol_filter(data, window_size, poly_order)
```

Листинг 3: Savitzky-Golay filter implementation (Python)

```r
savitzky_golay_filter <- function(data, window_size = 11, poly_order = 3) {
  """
  Apply Savitzky-Golay filter to data

  Parameters:
  data: vector - input data
  window_size: integer - window size (must be odd)
  poly_order: integer - polynomial order

  Returns:
  filtered_data: vector - filtered data
  """
  if (length(data) < window_size) {
    window_size <- ifelse(length(data) %% 2 == 1, length(data), length(data) - 1)
  }

  sgolayfilt(data, p = poly_order, n = window_size)
```

```
}
```

Листинг 4: Savitzky-Golay filter implementation (R)

Функция нормализации IQR

```python
def iqr_normalization(data):
    """
    Normalize data using IQR method

    Parameters:
    data: array-like - input data

    Returns:
    normalized_data: array - normalized data
    """
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    median = np.median(data)

    # Protection against division by zero
    if IQR == 0:
        return data - median
    else:
        return (data - median) / IQR
```

Листинг 5: IQR normalization implementation (Python)

```r
iqr_normalization <- function(data) {
  """
  Normalize data using IQR method

  Parameters:
  data: vector - input data

  Returns:
  normalized_data: vector - normalized data
  """
  Q1 <- quantile(data, 0.25, na.rm = TRUE)
  Q3 <- quantile(data, 0.75, na.rm = TRUE)
  IQR <- Q3 - Q1
  med <- median(data, na.rm = TRUE)

  # Protection against division by zero
  if (IQR == 0) {
    return(data - med)
  } else {
    return((data - med) / IQR)
  }
}
```

Листинг 6: IQR normalization implementation (R)

Функция создания пропусков

```python
def introduce_missing_values(data, missing_percentage=0.2, random_state=42):
    """
    Create missing values in data

    Parameters:
    data: DataFrame - input data
    missing_percentage: float - percentage of rows with missing values (0-1)
    random_state: int - for reproducibility

    Returns:
    data_with_missing: DataFrame - data with missing values
    """
```

```python
    np.random.seed(random_state)
    data_with_missing = data.copy()

    # Select rows for missing values
    n_rows = int(len(data) * missing_percentage)
    rows_with_missing = np.random.choice(data.index, n_rows, replace=False)

    for idx in rows_with_missing:
        # Select random columns for missing values
        n_missing = np.random.randint(1, len(factors)//2 + 1)
        cols_to_miss = np.random.choice(factors, n_missing, replace=False)
        data_with_missing.loc[idx, cols_to_miss] = np.nan

    return data_with_missing
```

Листинг 7: Missing values generation (Python)

```r
introduce_missing_values <- function(data, missing_percentage = 0.2, seed = 42) {
  """
  Create missing values in data

  Parameters:
  data: data.frame - input data
  missing_percentage: numeric - percentage of rows with missing values (0-1)
  seed: integer - for reproducibility

  Returns:
  data_with_missing: data.frame - data with missing values
  """
  set.seed(seed)
  data_with_missing <- data

  n_rows <- round(nrow(data) * missing_percentage)
  rows_with_missing <- sample(1:nrow(data), n_rows)

  for (i in rows_with_missing) {
    n_missing <- sample(1:(length(factors)%/%2 + 1), 1)
    cols_to_miss <- sample(factors, n_missing)
    data_with_missing[i, cols_to_miss] <- NA
  }

  return(data_with_missing)
}
```

Листинг 8: Missing values generation (R)

Функция восстановления пропусков

```python
def impute_missing_values(data, method='mean', window_size=5):
    """
    Impute missing values

    Parameters:
    data: DataFrame - data with missing values
    method: str - imputation method ('mean', 'moving_average')
    window_size: int - window size for moving average

    Returns:
    imputed_data: DataFrame - data with imputed values
    """
    data_imputed = data.copy()

    for col in data.columns:
        if data[col].isna().any():
            if method == 'mean':
                # Fill with column mean
                data_imputed[col] = data[col].fillna(data[col].mean())
            elif method == 'moving_average':
                # Fill with moving average
                data_imputed[col] = data[col].fillna(
                    data[col].rolling(window=window_size, min_periods=1).mean()
```

```
                )

    return data_imputed
```

Листинг 9: Missing values imputation (Python)

```
impute_missing_values <- function(data, method = "mean", window_size = 5) {
  """
  Impute missing values

  Parameters:
  data: data.frame - data with missing values
  method: character - imputation method ('mean', 'moving_average')
  window_size: integer - window size for moving average

  Returns:
  imputed_data: data.frame - data with imputed values
  """
  data_imputed <- data

  for (col in names(data)) {
    if (any(is.na(data[[col]]))) {
      if (method == "mean") {
        data_imputed[[col]] <- ifelse(is.na(data[[col]]),
                                      mean(data[[col]], na.rm = TRUE),
                                      data[[col]])
      } else if (method == "moving_average") {
        # Use zoo for moving average
        ma <- rollapply(data[[col]], window_size, mean, na.rm = TRUE,
                        fill = NA, align = "right", partial = TRUE)
        data_imputed[[col]] <- ifelse(is.na(data[[col]]), ma, data[[col]])
      }
    }
  }

  return(data_imputed)
}
```

Листинг 10: Missing values imputation (R)

Основной код выполнения

```
# Load and prepare data
# data = pd.read_csv('your_dataset.csv')
# factors = ['feature1', 'feature2', 'feature3', ...]  # feature list
# target = 'target_variable'  # target variable

# Assume data is already loaded and prepared
# data contains factors and target

print("Initial data size:", data.shape)
print("Number of factors:", len(factors))

# Apply Savitzky-Golay filter to factors
print("Applying Savitzky-Golay filter...")
for col in factors:
    data[col + '_filtered'] = savitzky_golay_filter(data[col].values)

# Normalize factors using IQR method
print("Applying IQR normalization...")
for col in factors:
    data[col + '_normalized'] = iqr_normalization(data[col].values)

# Create missing values in data
print("Creating missing values...")
data_missing = introduce_missing_values(data[factors + [target]], 0.2)
print("Number of missing values after creation:", data_missing.isna().sum().sum())

# Impute missing values using two methods
print("Imputing missing values with mean method...")
data_imputed_mean = impute_missing_values(data_missing, 'mean')
```

```python

print("Imputing missing values with moving average method...")
data_imputed_ma = impute_missing_values(data_missing, 'moving_average', 5)

# Function for training and evaluating model
def train_and_evaluate(X, y, test_size=0.2, random_state=42):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    return model, mse, r2, mae, model.coef_

# Train on original data
print("Training on original data...")
model_orig, mse_orig, r2_orig, mae_orig, coef_orig = train_and_evaluate(
    data[factors], data[target]
)

# Train on data imputed with mean
print("Training on mean-imputed data...")
model_mean, mse_mean, r2_mean, mae_mean, coef_mean = train_and_evaluate(
    data_imputed_mean[factors], data_imputed_mean[target]
)

# Train on data imputed with moving average
print("Training on moving average-imputed data...")
model_ma, mse_ma, r2_ma, mae_ma, coef_ma = train_and_evaluate(
    data_imputed_ma[factors], data_imputed_ma[target]
)

# Compare results
results = pd.DataFrame({
    'Model': ['Original', 'Mean', 'Moving Average'],
    'MSE': [mse_orig, mse_mean, mse_ma],
    'R  ': [r2_orig, r2_mean, r2_ma],
    'MAE': [mae_orig, mae_mean, mae_ma]
})

print("Model comparison:")
print(results)

# Analyze coefficient differences
coef_comparison = pd.DataFrame({
    'Factor': factors,
    'Original': coef_orig,
    'Difference_Mean': coef_mean - coef_orig,
    'Difference_Moving_Average': coef_ma - coef_orig
})

print("\nModel coefficient differences:")
print(coef_comparison)
```

Листинг 11: Main analysis process (Python)

```r
# Load and prepare data
# data <- read.csv('your_dataset.csv')
# factors <- c('feature1', 'feature2', 'feature3', ...)  # feature list
# target <- 'target_variable'  # target variable

# Assume data is already loaded and prepared
# data contains factors and target

cat("Initial data size:", dim(data), "\n")
cat("Number of factors:", length(factors), "\n")

```

```r
# Apply Savitzky-Golay filter to factors
cat("Applying Savitzky-Golay filter...\n")
for (col in factors) {
  data[[paste0(col, "_filtered")]] <- savitzky_golay_filter(data[[col]])
}

# Normalize factors using IQR method
cat("Applying IQR normalization...\n")
for (col in factors) {
  data[[paste0(col, "_normalized")]] <- iqr_normalization(data[[col]])
}

# Create missing values in data
cat("Creating missing values...\n")
data_missing <- introduce_missing_values(data[c(factors, target)], 0.2)
cat("Number of missing values after creation:", sum(is.na(data_missing)), "\n")

# Impute missing values using two methods
cat("Imputing missing values with mean method...\n")
data_imputed_mean <- impute_missing_values(data_missing, "mean")

cat("Imputing missing values with moving average method...\n")
data_imputed_ma <- impute_missing_values(data_missing, "moving_average", 5)

# Function for training and evaluating model
train_and_evaluate <- function(X, y, test_size = 0.2, seed = 42) {
  set.seed(seed)
  train_index <- createDataPartition(y, p = 1 - test_size, list = FALSE)

  X_train <- X[train_index, ]
  X_test <- X[-train_index, ]
  y_train <- y[train_index]
  y_test <- y[-train_index]

  model <- lm(y_train ~ ., data = data.frame(X_train))
  y_pred <- predict(model, newdata = data.frame(X_test))

  mse <- mean((y_test - y_pred)^2)
  r2 <- cor(y_test, y_pred)^2
  mae <- mean(abs(y_test - y_pred))

  return(list(model = model, mse = mse, r2 = r2, mae = mae, coef = coef(model)))
}

# Train on original data
cat("Training on original data...\n")
result_orig <- train_and_evaluate(data[factors], data[[target]])

# Train on data imputed with mean
cat("Training on mean-imputed data...\n")
result_mean <- train_and_evaluate(data_imputed_mean[factors],
                                  data_imputed_mean[[target]])

# Train on data imputed with moving average
cat("Training on moving average-imputed data...\n")
result_ma <- train_and_evaluate(data_imputed_ma[factors],
                                data_imputed_ma[[target]])

# Compare results
results <- data.frame(
  Model = c("Original", "Mean", "Moving Average"),
  MSE = c(result_orig$mse, result_mean$mse, result_ma$mse),
  R2 = c(result_orig$r2, result_mean$r2, result_ma$r2),
  MAE = c(result_orig$mae, result_mean$mae, result_ma$mae)
)

cat("Model comparison:\n")
print(results)

# Analyze coefficient differences
coef_comparison <- data.frame(
  Factor = factors,
  Original = result_orig$coef[-1],  # exclude intercept
```

```
  Difference_Mean = result_mean$coef[-1] - result_orig$coef[-1],
  Difference_Moving_Average = result_ma$coef[-1] - result_orig$coef[-1]
)

cat("Model coefficient differences:\n")
print(coef_comparison)
```

Листинг 12: Main analysis process (R)

Визуализация результатов

```
# Quality metrics visualization
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

metrics = ['MSE', 'R  ', 'MAE']
values = [results['MSE'].values, results['R  '].values, results['MAE'].values]
colors = ['lightcoral', 'lightgreen', 'lightblue']
titles = ['MSE Model Comparison', 'R   Model Comparison', 'MAE Model Comparison']
ylabels = ['Mean Squared Error', 'R-squared', 'Mean Absolute Error']

for i, (ax, metric, value, color, title, ylabel) in enumerate(
    zip(axes, metrics, values, colors, titles, ylabels)):

    bars = ax.bar(results['Model'], value, color=color, alpha=0.7)
    ax.set_title(title, fontsize=12)
    ax.set_ylabel(ylabel, fontsize=10)
    ax.tick_params(axis='x', rotation=45)

    # Add values on bars
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height + 0.01 * max(value),
                f'{height:.3f}', ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()

# Coefficient differences visualization
plt.figure(figsize=(12, 6))
x_pos = np.arange(len(factors))
width = 0.35

plt.bar(x_pos - width/2, coef_comparison['Difference_Mean'],
        width, label='Difference (Mean)', alpha=0.7, color='orange')
plt.bar(x_pos + width/2, coef_comparison['Difference_Moving_Average'],
        width, label='Difference (Moving Average)', alpha=0.7, color='purple')

plt.xlabel('Factors', fontsize=12)
plt.ylabel('Coefficient Differences', fontsize=12)
plt.title('Model Coefficient Differences Compared to Original', fontsize=14)
plt.xticks(x_pos, factors, rotation=45, ha='right')
plt.legend()
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

# Save results
results.to_csv('results_comparison.csv', index=False, encoding='utf-8')
coef_comparison.to_csv('coefficients_comparison.csv', index=False, encoding='utf-8')
print("Results saved to CSV files")
```

Листинг 13: Model comparison visualization (Python)

```
# Quality metrics visualization
library(ggplot2)
library(gridExtra)

# Prepare data for visualization
results_long <- results %>%
  pivot_longer(cols = c(MSE, R2, MAE),
               names_to = "Metric", values_to = "Value")
```

```r

# Plots for each metric
p1 <- ggplot(results, aes(x = Model, y = MSE, fill = Model)) +
  geom_bar(stat = "identity", alpha = 0.7) +
  geom_text(aes(label = round(MSE, 3)), vjust = -0.5) +
  labs(title = "MSE Model Comparison", y = "Mean Squered Error") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p2 <- ggplot(results, aes(x = Model, y = R2, fill = Model)) +
  geom_bar(stat = "identity", alpha = 0.7) +
  geom_text(aes(label = round(R2, 3)), vjust = -0.5) +
  labs(title = "R   Model Comparison", y = "R-squared") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p3 <- ggplot(results, aes(x = Model, y = MAE, fill = Model)) +
  geom_bar(stat = "identity", alpha = 0.7) +
  geom_text(aes(label = round(MAE, 3)), vjust = -0.5) +
  labs(title = "MAE Model Comparison", y = "Mean Absolute Error") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Combined visualization
grid.arrange(p1, p2, p3, ncol = 3)

# Coefficient differences visualization
coef_long <- coef_comparison %>%
  pivot_longer(cols = c(Difference_Mean, Difference_Moving_Average),
               names_to = "Method", values_to = "Difference")

ggplot(coef_long, aes(x = Factor, y = Difference, fill = Method)) +
  geom_bar(stat = "identity", position = "dodge", alpha = 0.7) +
  labs(title = "Model Coefficient Differences Compared to Original",
       x = "Factors", y = "Coefficient Differences") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_manual(values = c("orange", "purple"))

# Save results
write.csv(results, "results_comparison.csv", row.names = FALSE, fileEncoding = "UTF-8")
write.csv(coef_comparison, "coefficients_comparison.csv", row.names = FALSE,
    fileEncoding = "UTF-8")
cat("Results saved to CSV files\n")
```

Листинг 14: Model comparison visualization (R)