РК-2

Трифонов Дмитрий Алексеевич, ИУ5-65Б, Вариант 18

Задание

- Дан набор данных: https://www.kaggle.com/rhuebner/human-resources-data-set
- Для этого набора данных необходимо построить модель классификации (регрессии).
- Необходимо использовать метод опорных векторов и градиентный бустинг.
- Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик).
- Какие метрики качества Вы использовали и почему?
- Какие выводы Вы можете сделать о качестве построенных моделей?
- Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

Импорт зависимостей

In [ ]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from datetime import datetime
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2
import seaborn as sns
import matplotlib.pyplot as plt
```

Первичный анализ данных

In [ ]:
```
df = pd.read_csv("HRDataset_v14.csv")
```

In [ ]:
```
df.head()
```

Out[ ]:

	Employee_Name	EmpID	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptI
0	Adinolfi, Wilson K	10026	0	0	1	1	
1	Ait Sidi, Karthikeyan	10084	1	1	1	5	
2	Akinkuolie, Sarah	10196	1	1	0	5	
3	Alagbe,Trina	10088	1	1	0	1	
4	Anderson, Carol	10069	0	2	0	5	

5 rows × 36 columns

In [ ]:
```
df.columns
```

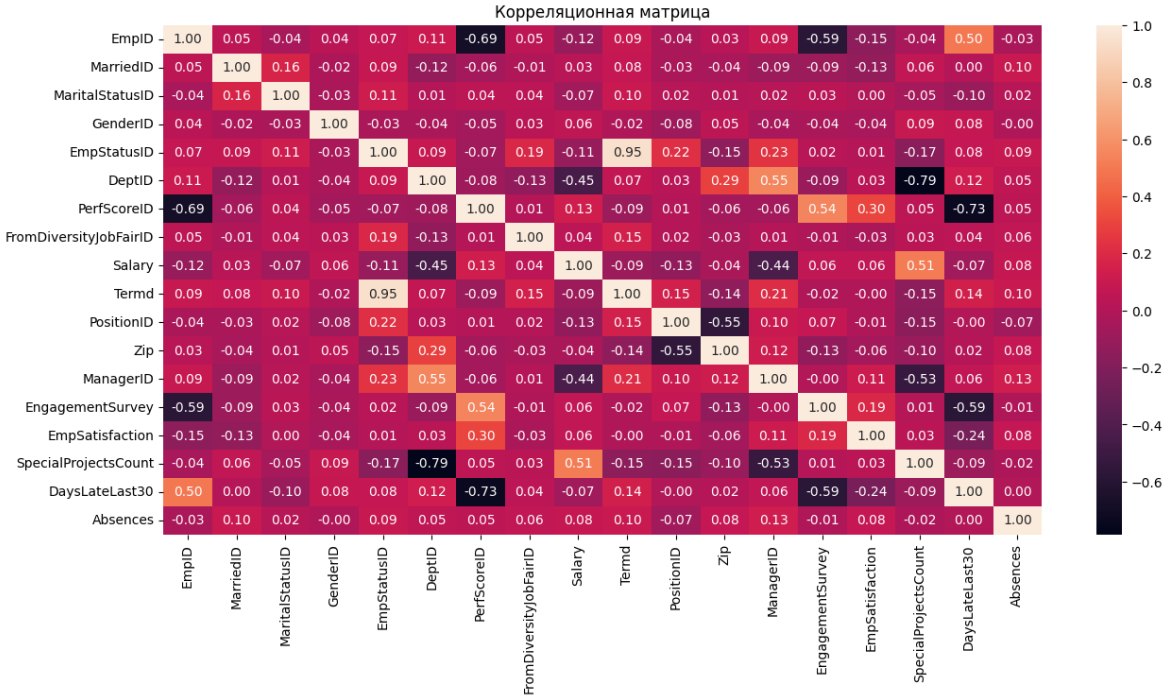
Out[ ]:
```
Index(['Employee_Name', 'EmpID', 'MarriedID', 'MaritalStatusID', 'Gende
rID',
       'EmpStatusID', 'DeptID', 'PerfScoreID', 'FromDiversityJobFairI
D',
       'Salary', 'Termd', 'PositionID', 'Position', 'State', 'Zip', 'DO
B',
       'Sex', 'MaritalDesc', 'CitizenDesc', 'HispanicLatino', 'RaceDes
c',
       'DateofHire', 'DateofTermination', 'TermReason', 'EmploymentStat
us',
       'Department', 'ManagerName', 'ManagerID', 'RecruitmentSource',
       'PerformanceScore', 'EngagementSurvey', 'EmpSatisfaction',
       'SpecialProjectsCount', 'LastPerformanceReview_Date', 'DaysLateL
ast30',
       'Absences'],
      dtype='object')
```

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Employee_Name                object
EmpID                         int64
MarriedID                     int64
MaritalStatusID               int64
GenderID                      int64
EmpStatusID                   int64
DeptID                        int64
PerfScoreID                   int64
FromDiversityJobFairID        int64
Salary                        int64
Termd                         int64
PositionID                    int64
Position                     object
State                        object
Zip                           int64
DOB                          object
Sex                          object
MaritalDesc                  object
CitizenDesc                  object
HispanicLatino               object
RaceDesc                     object
DateofHire                   object
DateofTermination            object
TermReason                   object
EmploymentStatus             object
Department                   object
ManagerName                  object
ManagerID                   float64
RecruitmentSource            object
PerformanceScore             object
EngagementSurvey            float64
EmpSatisfaction               int64
SpecialProjectsCount          int64
LastPerformanceReview_Date   object
DaysLateLast30                int64
Absences                      int64
dtype: object
```

Корреляционная матрица

In [ ]:
```
fig, ax = plt.subplots(figsize=(15,7))
sns.heatmap(df.corr(), annot=True, fmt='.2f')
ax.set_title('Корреляционная матрица')
plt.show()
```



Вывод: Будем решать задачу регрессии для признака `Salary` (предсказание зарплаты сотрудника по его характеристикам)

Обработка пропусков

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Employee_Name                  0
EmpID                          0
MarriedID                      0
MaritalStatusID                0
GenderID                       0
EmpStatusID                    0
DeptID                         0
PerfScoreID                    0
FromDiversityJobFairID         0
Salary                         0
Termd                          0
PositionID                     0
Position                       0
State                          0
Zip                            0
DOB                            0
Sex                            0
MaritalDesc                    0
CitizenDesc                    0
HispanicLatino                 0
RaceDesc                       0
DateofHire                     0
DateofTermination            207
TermReason                     0
EmploymentStatus               0
Department                     0
ManagerName                    0
ManagerID                      8
RecruitmentSource              0
PerformanceScore               0
EngagementSurvey               0
EmpSatisfaction                0
SpecialProjectsCount           0
LastPerformanceReview_Date     0
DaysLateLast30                 0
Absences                       0
dtype: int64
```

Единственная колонка с пропусками - `DateOfTermination`, но её мы не будем использовать для обучения.

Поэтому можно считать, что в исходном датасете пропусков нет.

Удаление ненужных колонок

In [ ]:
```
# normalize:
# categorize:
df = df.drop(columns=['Employee_Name', 'EmpID', 'Position', 'Zip', 'Sex
                      'MaritalDesc', 'CitizenDesc', 'LastPerformanceRev
                      'ManagerName', 'EmploymentStatus', 'TermReason',
                      'DateofTermination', 'DateofHire', 'ManagerID'])
```

Колонку `DOB` (Date of Birth) заменим на соответствующую ей колонку `Age`

In [ ]:
```
df['DOB'] = pd.to_datetime(df['DOB'])

current_date = datetime.now()
```

```
df['Age'] = df['DOB'].apply(lambda x: current_date.year - x.year - ((cu
df.drop(columns=['DOB'], inplace=True)
```

```
C:\Users\Dmitriy\AppData\Local\Temp\ipykernel_7884\2028958031.py:1: User
Warning: Could not infer format, so each element will be parsed individu
ally, falling back to `dateutil`. To ensure parsing is consistent and as
-expected, please specify a format.
  df['DOB'] = pd.to_datetime(df['DOB'])
```

Кодирование признаков

In [ ]:
```
label_encoder = LabelEncoder()

for col_name in ['State', 'PerformanceScore', 'RecruitmentSource', 'Dep
    df[col_name] = label_encoder.fit_transform(df[col_name])

df.head()
```

Out[ ]:

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiver
0	0.0	0.00	1.0	0.0	0.8	1.000000	
1	1.0	0.25	1.0	1.0	0.4	0.666667	
2	1.0	0.25	0.0	1.0	0.8	0.666667	
3	1.0	0.25	0.0	0.0	0.8	0.666667	
4	0.0	0.50	0.0	1.0	0.8	0.666667	

5 rows × 22 columns

Масштабирование признаков

In [ ]:
```
df_scaler = MinMaxScaler()

df_columns = df.columns.to_list()
df_columns.remove('Salary')

for col_name in df_columns:
    df[[col_name]] = df_scaler.fit_transform(df[[col_name]])

df.head()
```

Out[ ]:

	MarriedID	MaritalStatusID	GenderID	EmpStatusID	DeptID	PerfScoreID	FromDiver
0	0.0	0.00	1.0	0.0	0.8	1.000000	
1	1.0	0.25	1.0	1.0	0.4	0.666667	
2	1.0	0.25	0.0	1.0	0.8	0.666667	
3	1.0	0.25	0.0	0.0	0.8	0.666667	
4	0.0	0.50	0.0	1.0	0.8	0.666667	

5 rows × 22 columns

Выбор метрик

Mean absolute error - средняя абсолютная ошибка

$$MAE(y, \hat{y}) = \frac{1}{N}\Sigma_{i=1}^{N}|y_i - \hat{y}_i|$$

где:

y - истинное значение целевого признака

$\hat{y}$- предсказанное значение целевого признака

N- размер тестовой выборки

Чем ближе значение к нулю, тем лучше качество регрессии.

Основная проблема метрики состоит в том, что она не нормирована.

Вычисляется с помощью функции mean_absolute_error.

Mean squared error - средняя квадратичная ошибка

$$MSE(y, \hat{y}) = \frac{1}{N}\Sigma_{i=1}^{N}(y_i - \hat{y}_i)2$$

где:

y- истинное значение целевого признака

$\hat{y}$- предсказанное значение целевого признака

N- размер тестовой выборки

Вычисляется с помощью функции mean_squared_error.

Метрика R2 или коэффициент детерминации

$$R^2(y, \hat{y}) = 1 - \frac{\Sigma_{i=1}^{N}(y_i - \hat{y_i^2})}{\Sigma_{i=1}^{N}(y_i - \hat{y_i^2})}$$

где:

y - истинное значение целевого признака

$\hat{y}$- предсказанное значение целевого признака

N- размер тестовой выборки

$$\overline{y_i} = \frac{1}{N} \cdot \Sigma_{i=1}^{N} y_i$$

Вычисляется с помощью функции r2_score.

Метрика R2 показывает относительное отклонение предсказанных значений от реальных, в то время как MAE и MSE показывают ошибку в единицах измерения целевого признака. Так как метрика R2 в одиночку достаточно неточная, MAE и MSE гармонично её дополняют и в совокупности дают полную картину о точности модели.

Разбиение датасета на обучающую и тестовую выборки

In [ ]:
```
x_df = df.drop(columns='Salary')
y_df = df['Salary']

x_train, x_test, y_train, y_test = train_test_split(x_df, y_df, test_si

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(233, 21) (233,)
(78, 21) (78,)
```

Обучение моделей

In [ ]:
```
svr  = SVR()
gbr = GradientBoostingRegressor()
```

In [ ]:
```
svr.fit(x_train, y_train)
```

Out[ ]:
```
SVR
SVR()
```

In [ ]:
```
gbr.fit(x_train, y_train)
```

Out[ ]:
```
GradientBoostingRegressor
GradientBoostingRegressor()
```

Получение предсказаний от моделей

In [ ]:
```
svr_predict = svr.predict(x_test)
gbr_predict = gbr.predict(x_test)
```

Оценка качества моделей

In [ ]:
```
print('Показатели модели опорных векторов:')
print('MAE: {}, MSE: {}, R^2: {}'.format(mean_absolute_error(y_test, sv
```

```
Показатели модели опорных векторов:
MAE: 12362.747063462844, MSE: 358378461.59747463, R^2: -0.04544331214857
0225
```

In [ ]:
```
print('Показатели модели градиентного бустинга:')
print('MAE: {}, MSE: {}, R^2: {}'.format(mean_absolute_error(y_test, gb
```

```
Показатели модели градиентного бустинга:
MAE: 9033.43305105141, MSE: 196777528.3863341, R^2: 0.42597066208828205
```

Вывод

Как видим, модели получились не очень качественные (большая MAE), что связанно с маленьким объёмом данных (около 300 строк)

Более точная оказалась модель градиентного бустинга.