In [1]:
```
import numpy as np
import scipy.stats as sps
import pandas as pd
from tqdm.notebook import tqdm
import warnings

import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
import statsmodels.api as sm
import statsmodels.datasets as smd

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')
sns.set(style='darkgrid', font_scale=1.7, palette='Set2')
```

Обобщенная линейная модель

Ожидаемый отклик $y = \mu_\theta(x)$, где $g(\mu_\theta(x)) = x^T\theta$.

Наблюдаемый отклик $Y_i \sim \mathsf{P}_{\mu_\theta(x_i)}$.

Оценка ожидаемого отклика $\widehat{y} = g^{-1}\left(x^T\widehat{\theta}\right)$.

Документация (https://www.statsmodels.org/stable/generated/statsmodels.genmod.generalized_linear_model.GLM.html)

```
model = GLM(y_train, X_train, family=sm.family.Gaussian())
```

Методы класса:

- `reg.fit(X=X_train)` — возвращает объект типа `statsmodels.base.model.LikelihoodModelResults`

Аргументы:

- `X_train`, `Y_train` — выборка;
- `family` — семейство распределений из Списка (https://www.statsmodels.org/stable/glm.html#families).

Как и для гауссовской линейной модели glm можно задавать двумя способами:

- как обычно передав таргет и матрицу признаков (единичного признака нет).
- указав формулу в виде `target ~ features`, которая означает линейную функцию `target` от `features` (единичный признак есть).

Логистическая регрессия

В качестве линеризации ожидаемого отклика рассматриваем логит-функцию, а наблюдаемый отклик распределен из $Bern(\mu_\theta(x))$, где $\mu_\theta(x) = \dfrac{1}{1 + e^{x^T\theta}}$.

Возьмем датасет (https://vincentarelbundock.github.io/Rdatasets/doc/MASS/biopsy.html) о раке груди в котором по ряду признаков нужно оценить доброкачественная или злокачественная опухоль.

In [2]:
```
cancer = smd.get_rdataset("biopsy", package="MASS").data
cancer = cancer.set_index(["ID"])
cancer.head()
```

Out[2]:

ID	V1	V2	V3	V4	V5	V6	V7	V8	V9	class
1000025	5	1	1	1	2	1.0	3	1	1	benign
1002945	5	4	4	5	7	10.0	3	2	1	benign
1015425	3	1	1	1	2	2.0	3	1	1	benign
1016277	6	8	8	1	3	4.0	3	7	1	benign
1017023	4	1	1	3	2	1.0	3	1	1	benign

In [3]:
```
cancer.describe()
```

Out[3]:

	V1	V2	V3	V4	V5	V6	V7	V8	V
count	699.000000	699.000000	699.000000	699.000000	699.000000	683.000000	699.000000	699.000000	699.0000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.544656	3.437768	2.866953	1.5894
std	2.815741	3.051459	2.971913	2.855379	2.214300	3.643857	2.438364	3.053634	1.7150
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.0000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.0000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.0000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.0000

Удалим пропуски

In [4]:
```
cancer = cancer.dropna()
cancer.describe()
```

Out[4]:

	V1	V2	V3	V4	V5	V6	V7	V8	V
count	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.0000
mean	4.442167	3.150805	3.215227	2.830161	3.234261	3.544656	3.445095	2.869693	1.6032
std	2.820761	3.065145	2.988581	2.864562	2.223085	3.643857	2.449697	3.052666	1.7326
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.0000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.0000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.0000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.0000

Приведем класс к числовому

In [5]:
```
cancer['y'] = (cancer['class'] == 'benign') * 1
cancer = cancer.drop(["class"], axis=1)
cancer.head()
```

Out[5]:

ID	V1	V2	V3	V4	V5	V6	V7	V8	V9	y
1000025	5	1	1	1	2	1.0	3	1	1	1
1002945	5	4	4	5	7	10.0	3	2	1	1
1015425	3	1	1	1	2	2.0	3	1	1	1
1016277	6	8	8	1	3	4.0	3	7	1	1
1017023	4	1	1	3	2	1.0	3	1	1	1

Разделим выборку на обучающую и тестовую части

In [6]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(cancer, test_size=0.1, random_state=19)
```

Определим модель — линейную комбинацию всех признаков. Обучим GLM, используя биномиальное семейство, что соответствует логистической регрессии.

In [7]:
```
all_params = "+".join(cancer.columns.difference(["y"]))
glm_model = smf.glm("y ~ " + all_params, train, family=sm.families.Binomial())
glm_results = glm_model.fit()
```

Посмотрим статистические свойства обученной модели

In [8]:
```
print(glm_results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  614
Model:                            GLM   Df Residuals:                      604
Model Family:                Binomial   Df Model:                            9
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -47.091
Date:                Tue, 29 Nov 2022   Deviance:                       94.182
Time:                        22:45:56   Pearson chi2:                     436.
No. Iterations:                     8   Pseudo R-squ. (CS):             0.6807
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      9.9786      1.193      8.365      0.000       7.641      12.317
V1            -0.5141      0.139     -3.691      0.000      -0.787      -0.241
V2            -0.0452      0.213     -0.212      0.832      -0.463       0.373
V3            -0.2963      0.224     -1.325      0.185      -0.734       0.142
V4            -0.3049      0.124     -2.454      0.014      -0.548      -0.061
V5            -0.0342      0.186     -0.184      0.854      -0.399       0.331
V6            -0.4042      0.097     -4.161      0.000      -0.595      -0.214
V7            -0.4790      0.173     -2.772      0.006      -0.818      -0.140
V8            -0.1441      0.138     -1.042      0.297      -0.415       0.127
V9            -0.5853      0.310     -1.891      0.059      -1.192       0.021
==============================================================================
```

Посчитаем точность предсказаний модели на обучении и на тесте

In [10]:
```
print(accuracy_score(train['y'], glm_results.predict() > 0.5))
print(accuracy_score(test['y'], glm_results.predict(test) > 0.5))
```

```
0.9690553745928339
0.9565217391304348
```

Оставим только стат. значимые признаки и заново обучим модель

In [11]:
```
glm_model = smf.glm("y ~ V1 + V4 + V6 + V7", train, family=sm.families.Binomial())
glm_results = glm_model.fit()
print(glm_results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  614
Model:                            GLM   Df Residuals:                      609
Model Family:                Binomial   Df Model:                            4
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -56.822
Date:                Tue, 29 Nov 2022   Deviance:                       113.64
Time:                        22:46:03   Pearson chi2:                     814.
No. Iterations:                     8   Pseudo R-squ. (CS):             0.6704
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      9.9775      1.059      9.418      0.000       7.901      12.054
V1            -0.7790      0.125     -6.256      0.000      -1.023      -0.535
V4            -0.3859      0.115     -3.344      0.001      -0.612      -0.160
V6            -0.4852      0.089     -5.463      0.000      -0.659      -0.311
V7            -0.6981      0.150     -4.647      0.000      -0.993      -0.404
==============================================================================
```

Снова посмотрим на точность. На тесте она не изменилась

In [12]:
```
print(accuracy_score(train['y'], glm_results.predict() > 0.5))
print(accuracy_score(test['y'], glm_results.predict(test) > 0.5))
```

```
0.9625407166123778
0.9565217391304348
```

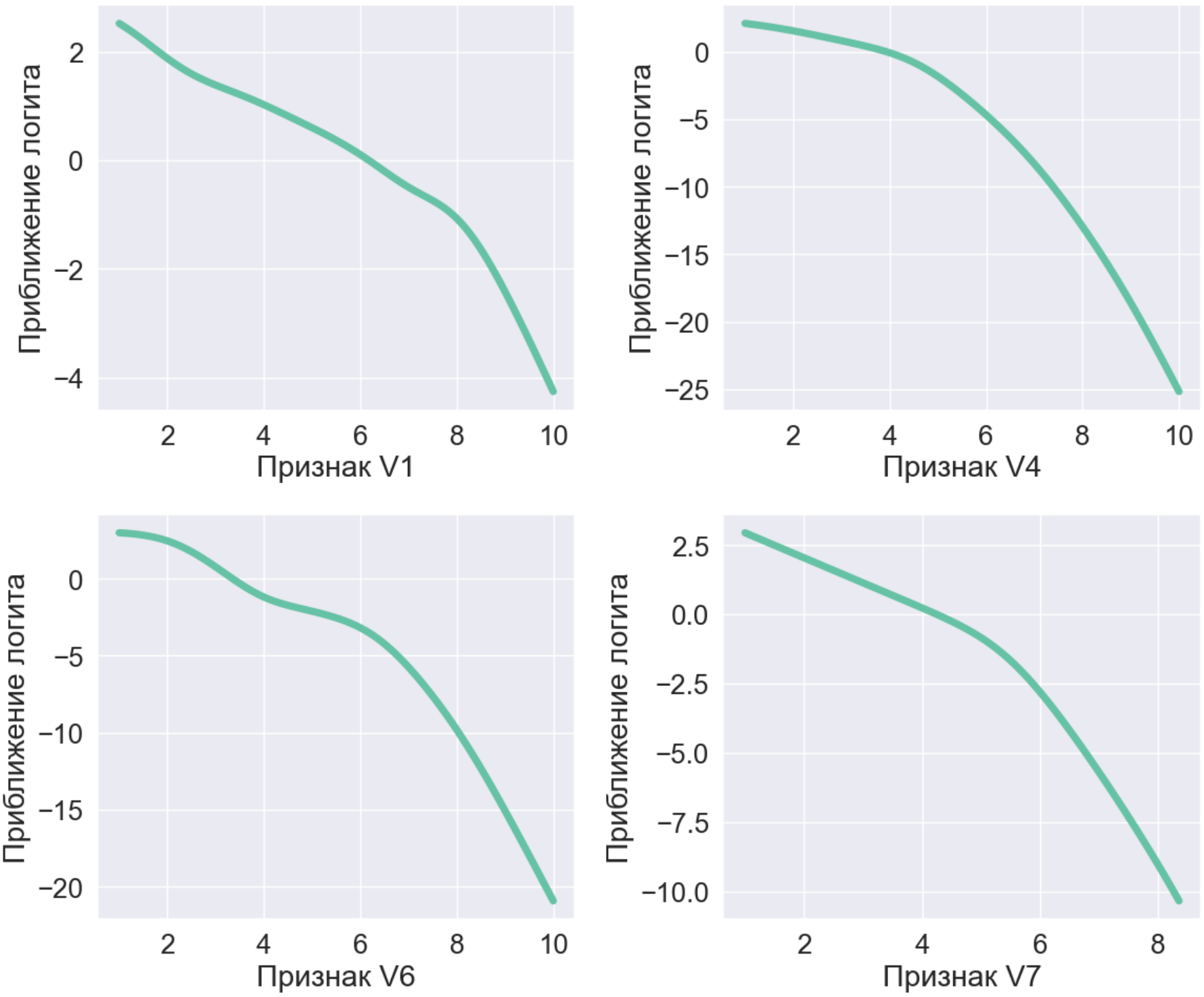
Для корректного применения статистических свойств логистической регрессии нужно выполнить проверку на линейность логита. Выполним ее

In [13]:
```python
h = 1  # ширина ядра
size = 100 # размер сетки

X = train[glm_results.params.index[1:]].values
y = train["y"]
plt.figure(figsize=(12, 10))
# цикл по признакам
for feature_idx in range(X.shape[1]):
    # отсортированная сетка по признаку feature_idx
    # для построения ядерной оценки
    x_grid = np.linspace(
        np.percentile(X[:, feature_idx], 5),
        np.percentile(X[:, feature_idx], 95),
        size
    )
    # гауссовское ядро с шириной h
    kernel = sps.norm(scale=h)
    # значения ядра в точках выборки
    kernel_values = kernel.pdf(X[:size, feature_idx][:size, np.newaxis] - x_grid[np.newaxis
    # оценка по ядерной регрессии для признака feature_idx
    y_est = (y[:size, np.newaxis] * kernel_values).sum(axis=0) / kernel_values.sum(axis=0)
    # приближение логита по оценкам y_est
    l_sm = np.log(y_est / (1 - y_est))

    # отрисуем графики
    plt.subplot(2, 2, feature_idx + 1)
    plt.plot(x_grid, l_sm, lw=5)
    plt.xlabel(f'Признак {glm_results.params.index[feature_idx + 1]}')
    plt.ylabel('Приближение логита')

plt.tight_layout()
plt.show()
```



Определим функцию, которая строит доверительный интервал для предсказаний

In [14]:
```python
def conf_int(model_results, data0, alpha=0.95):
    '''
    model_results - модель
    data0 - pd.DataFrame или pd.Slice с параметрами
    alpha - уровень значимости
    '''

    I = -model_results.model.hessian(model_results.params)
    y_hat = model_results.predict(data0, linear=True)

    x_names = model_results.params.index
    intercept = False

    if (x_names[0] == 'Intercept'):
        intercept = True
        x_names = x_names[1:]

    x0 = data0[x_names].values
    if len(x0.shape) == 1:
        x0 = x0.reshape(1, -1)
    if intercept:
        x0 = np.concatenate([np.ones((x0.shape[0], 1)), x0], axis = -1)

    delta = sps.norm.ppf((1. + alpha) / 2.) * np.sqrt(x0 @ np.linalg.inv(I) @ x0.T)
    delta = np.diag(delta).reshape(-1)

    res = pd.DataFrame({
        'prediction' : model_results.model.family.fitted(y_hat),
        'left bound' : model_results.model.family.fitted(y_hat - delta),
        'right bound': model_results.model.family.fitted(y_hat + delta)
    })
    return res
```

In [15]:
```python
conf_int(glm_results, test)
```

Out[15]:

	prediction	left bound	right bound
0	0.990761	0.974876	0.996637
1	0.999024	0.995727	0.999778
2	0.999514	0.997571	0.999903
3	0.994995	0.984699	0.998374
4	0.001968	0.000335	0.011469
...	...	...	...
64	0.002313	0.000466	0.011409
65	0.980225	0.951124	0.992142
66	0.997876	0.992369	0.999411
67	0.999024	0.995727	0.999778
68	0.997697	0.991907	0.999347

69 rows × 3 columns

Теперь мы можем сказать, что для пациента номер 65 вероятность того, что опухоль доброкачественная, можно оценить 95%-доверительным интервалом (95.11%, 99.21%).