In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

In [2]:
```
from sklearn.model_selection import train_test_split
```

In [3]:
```
from IPython.display import display, Math
```

Pobierz Dane & Preprocessing

In [4]:
```
DATA_PATH = "dane_zad1.csv"
```

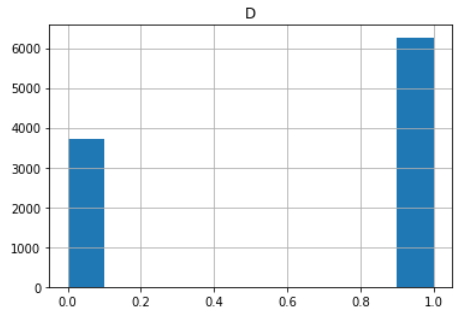
In [5]:
```
df = pd.read_csv(DATA_PATH)
```

In [6]:
```
df.head()
```

Out[6]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
0	20.092091	1.673548	9.923381	7.687586	9.121323	20.711181	23.347262	14.398792	10.460684	4.62344(
1	25.431807	5.625153	9.276870	27.683566	16.454241	13.341418	23.180162	10.874133	17.024978	20.59961:
2	23.647011	26.013198	4.729723	5.588910	13.077481	20.830146	1.883304	14.135294	4.313362	18.54297
3	21.197763	25.533612	3.673269	1.915619	25.069512	2.425438	25.536494	13.139628	26.144889	25.730316
4	18.426135	11.545287	20.044614	15.834745	10.729403	20.854713	1.145791	21.510991	22.230372	0.611073

In [7]:
```
df.hist(column="D");
```



Zbiór jest dość niezbalansowany, ale nie będę zajmował się tutaj resamplingiem.

In [8]:
```
data = df.values
X, y = data[:, :-1], data[:,-1]
```

Klasyfikatory

Zadanie polega na wyuczeniu poznanych klasyfikatorów (lista poniżej), przetestowaniu i dokonaniu oszacowania spodziewanego błędu klasyfikacji używając nierówności Hoeffdinga dla ewentualnych nowych przypadków.
Celem jest osiągnięcie jak najlepszej jakości klasyfikacji (jak najniższy spodziewany błąd generalizacji).
Należy to osiągnąć stosując metodę testowania za pomocą podziału próbki i dobierając właściwy rozmiar próbki testowej.
Rozwiązując zadanie nie należy stosować kroswalidacji.
W ramach zadania nie trzeba optymalizować wartości parametrów klasyfikatorów.

Lista poznanych klasyfikatorów:

> Naive Bayes;
>
> k-NN;
>
> Drzewo decyzyjne.

Walidator

Przyjmując poziom istotnosci rowny $\alpha = 0.9$, wyznaczę oszacowanie błędu generalizacji BG w zależności od wielkości zbioru testowego $|T|$, dlatego, że wielkość zbioru testowego związana jest z $\epsilon > 0$, zależnością $-ln(1-\alpha) = 2 * |T| * \epsilon$.
Wynika z tego, że ustalając T, przy wcześniej ustalonymy α, wyznaczony zostaje ϵ oraz błąd empiryczny BE_T.
Zadanie polega zatem na minimalizacji sumy błędu empirycznego BE_T i ϵ
$P(BG < BE_T + \epsilon) \geq 1 - e^{-2|T|\epsilon}$

In [9]:
```
def eps_value(alpha, test_size):
    return np.sqrt(-np.log(1 - alpha) / (2 * test_size))
```

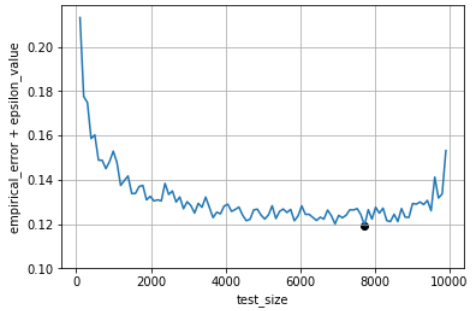
In [10]:
```
def err_plot(clf, alpha, k, to_cycle=5):
    bottom = 1/X.shape[0]
    dependency = []
    for test_freq in tqdm(np.linspace(bottom, 1-bottom, k+2)[1:-1]):
        upper_bound = 0
        for _ in range(to_cycle):
            x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=t
est_freq)
            sk_nb = clf.fit(x_train, y_train)
            upper_bound += 1-sk_nb.score(x_test, y_test) + eps_value(alpha, y_tes
t.shape[0])
        dependency.append((y_test.shape[0], upper_bound/to_cycle))

    dependency = np.array(dependency)
    upped_bound_idx = np.argmin(dependency[:, 1])
    test_size = dependency[upped_bound_idx, 0]
    ge_upper_bound = dependency[upped_bound_idx, 1]
    plt.grid(True)
    plt.xlabel("test_size")
    plt.ylabel("empirical_error + epsilon_value")
    plt.scatter(dependency[upped_bound_idx, 0], dependency[upped_bound_idx, 1], c
="k")
    plt.plot(dependency[:,0], dependency[:,1], )

    return int(test_size), eps_value(alpha, test_size), ge_upper_bound
```

Naive Bayes

In [11]:
```
from sklearn.naive_bayes import GaussianNB
```

In [12]:
```
test_size1, eps_value1, upper_bound1 = err_plot(clf=GaussianNB(), alpha=0.9, k=100)
```

```
100%|██████████| 100/100 [00:01<00:00, 64.49it/s]
```



In [13]:
```
display(Math(r'|T|: {} \\ \epsilon: {:.4f} \\ Błąd\ empiryczny: {:.4f} \\ Najmniejsze\ napotkane\ oszacowanie\ błędu\ generalizacji: {:.4f}'\
             .format(test_size1, eps_value1, upper_bound1-eps_value1, upper_bound1)))
```

$|T|$: 7723
ϵ : 0.0122
Błąd empiryczny : 0.1072
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1194

Użyte zostały defaultowe parametry:

In [14]:
```
GaussianNB().get_params()
```

Out[14]:
```
{'priors': None, 'var_smoothing': 1e-09}
```

k-NN

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [16]:
```
plt.figure(figsize=(10, 10))
for nbn in range(2, 25, 3):
    test_size2, eps_value2, upper_bound2 = err_plot(clf=KNeighborsClassifier(n_neighbors=nbn), alpha=0.9, k=100)
    display(Math(r'k: {} \\ |T|: {} \\ \epsilon: {:.4f} \\ Błąd\ empiryczny: {:.4f} \\ Najmniejsze\ napotkane\ oszacowanie\ błędu\ generalizacji: {:.4f}'\
                 .format(nbn, test_size2, eps_value2, upper_bound2-eps_value2, upper_bound2)))
plt.legend(range(2, 25, 3));
```

```
100%|██████████| 100/100 [02:10<00:00,  1.17s/it]
```

k : 2
$|T|$: 3070
ϵ : 0.0194
Błąd empiryczny : 0.1560
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1753

```
100%|██████████| 100/100 [02:39<00:00,  1.26s/it]
```

k : 5
$|T|$: 3466
ϵ : 0.0182
Błąd empiryczny : 0.1091
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1273

```
100%|██████████| 100/100 [02:54<00:00,  1.21s/it]
```

k : 8
$|T|$: 3367
ϵ : 0.0185
Błąd empiryczny : 0.0975
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1160

```
100%|██████████| 100/100 [02:58<00:00,  1.28s/it]
```

k : 11
$|T|$: 2575
ϵ : 0.0211
Błąd empiryczny : 0.0918
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1130

```
100%|██████████| 100/100 [03:09<00:00,  1.20s/it]
```

k : 14
$|T|$: 4060
ϵ : 0.0168
Błąd empiryczny : 0.0881
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1050

```
100%|██████████| 100/100 [03:11<00:00,  1.24s/it]
```

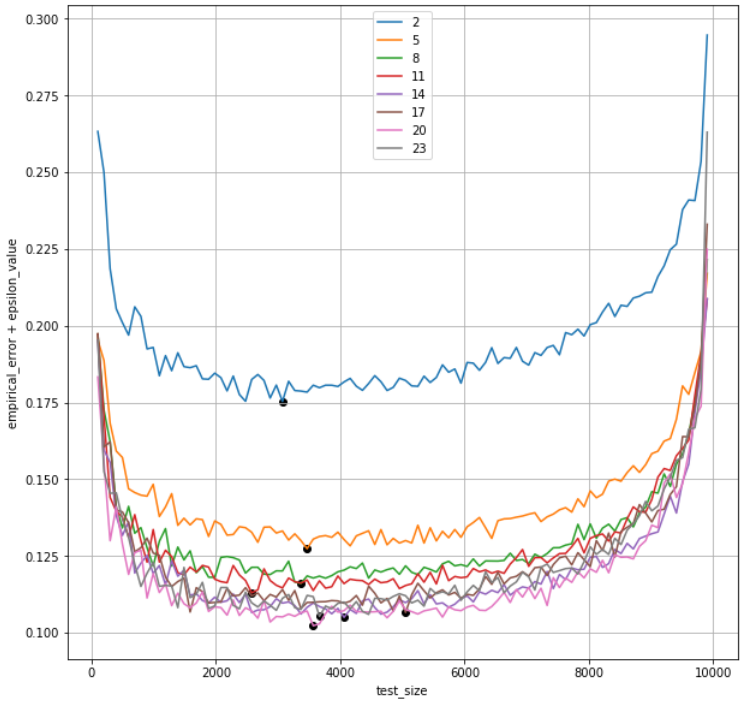
k : 17
$|T|$: 5050
ϵ : 0.0151
Błąd empiryczny : 0.0914
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1065

```
100%|██████████| 100/100 [03:12<00:00,  1.27s/it]
```

k : 20
$|T|$: 3565
ϵ : 0.0180
Błąd empiryczny : 0.0843
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1022

```
100%|██████████| 100/100 [03:17<00:00,  1.33s/it]
```

k : 23
$|T|$: 3664
ϵ : 0.0177
Błąd empiryczny : 0.0876
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1053



Użyte zostały defaultowe parametry, oprócz iterowanej liczby sąsiadów względem których podejmowana jest decyzja:

In [17]:
```
KNeighborsClassifier().get_params()
```

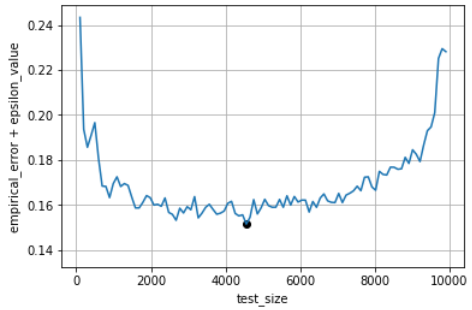
Out[17]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

Drzewo decyzyjne

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [19]:
```
test_size3, eps_value3, upper_bound3 = err_plot(clf=DecisionTreeClassifier(), alpha=0.9, k=100)
```

```
100%|██████████| 100/100 [00:19<00:00,  5.10it/s]
```



In [20]:
```
display(Math(r'|T|: {} \\ \epsilon: {:.4f} \\ Błąd\ empiryczny: {:.4f} \\ Najmniejsze\ napotkane\ oszacowanie\ błędu\ generalizacji: {:.4f}'\
             .format(test_size3, eps_value3, upper_bound3-eps_value3, upper_bound3)))
```

$|T|$: 4555
ϵ : 0.0159
Błąd empiryczny : 0.1356
Najmniejsze napotkane oszacowanie błędu generalizacji : 0.1515

Użyte zostały defaultowe parametry:

In [21]:
```
DecisionTreeClassifier().get_params()
```

Out[21]:
```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': False,
 'random_state': None,
 'splitter': 'best'}
```

Wnioski

Przy braku przeszukiwania przestrzeni hiperparametrów, badanych modeli, najlepsze wyniki uzyskał klasyfikator $k - NN$, przy $k = 20$.
Wyżej można przypatrzeć się wynikom tego modelu.

In [ ]:
```
```