МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП’ЮТЕРНИХ НАУК

Курсова робота з дисципліни

«Теорія розпізнавання образів»

Виконав: студент групи ІНм-91н
Тарасов Олександр Олексійович
Перевірив: Шелехов І. В.

СУМИ 2020

ЗМІСТ

Завдання 1

Постановка задачі

1. Реалізувати детерміноване вирішальне правило на основі методу побудови еталонів.

2. За допомогою побудованої правила розв’язати задачу розпізнавання двох стаціонарних за яскравістю зображень. Для цього на основі відповідних графічних файлів необхідно сформувати навчальну та тестову вибірки. Визначити параметри вирішального правила на навчальній вибірці.

3. Перевірити роботу класифікатора на тестових даних.

4. Результати роботи оформити звітом, який має містити: постановку задачі, навчальну вибірку даних у графічному виді , результати роботи на тестовій множині даних, всі параметри відтвореного класифікатора, вихідний код.

5. Визначити залежність ефективності вирішальних правил, сформованих за методом побудови еталонів, від кількості ознак розпізнавання та кількості класів розпізнавання.

Теоретичні відомості про метод еталонів

Для кожного класу за навчальною вибіркою формується еталон, що має такий вигляд:

$$\overline{x^0} = \left\{ x_1^0, x_2^0, ..., x_N^0 \right\}$$

де

$$x_i^0 = \frac{1}{K} \sum_{k=1}^{K} x_{ik}$$

K – кількість реалізацій образу в навчальній вибірці,

i - номер ознаки.

Власне кажучи, еталон – це усереднений за навчальною вибіркою абстрактний об’єкт (рис. 1). Абстрактним його називають, оскільки він може не співпадати не тільки ні з одною реалізацією навчальної вибірки, але і ні з однією реалізацією генеральної сукупності.

Розпізнавання виконується наступним чином. На вхід системи подається реалізація $\overline{x^*_i}$, належність якої до того або іншого образу системі невідома. Від цієї реалізації вимірюється відстань до еталонів всіх образів, і $\overline{x^*_i}$ система відносить до того образу, відстань до еталона якого мінімальна. Відстань вимірюється в тій метриці, що характерна для розв’язання певного класу задач розпізнавання.

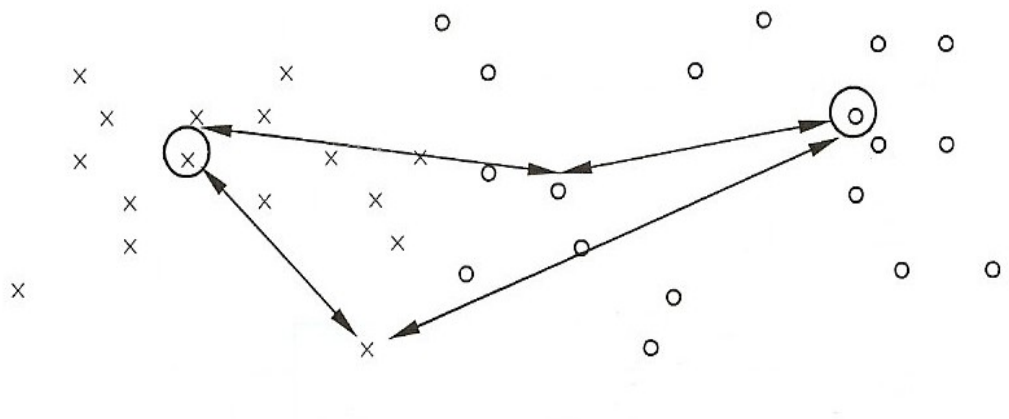


Рисунок 1 – Вирішальне правило «Мінімум відстані до еталона класу»

⊗ - еталон першого класу;
◎ - еталон другого класу.

Код реалізації та результати до завдання 1

Import libraries

In [1]:
```
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image

from labs.helpers import read_dataset
```

Prepare to load data

In [2]:
```
data_dir = "../../data/train"
classes = ["field", "road", "town", "water"]
file_pattern = "*{}*.jpg"
standard_shape = (50, 50, 3)
```

Load data and split it to train and test datasets

In [3]:
```
dataset, features = read_dataset(data_dir, classes, file_pattern, standa
train_df = dataset.sample(frac=0.85, random_state=18)
test_df = dataset.drop(train_df.index)
```

Demonstrate datasets

In [4]:
```
train_df.head()
```

Out[4]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:
36	36.0	72.0	72.0	39.0	75.0	75.0	42.0	78.0	78.0	44.0	...	39.0	74.
26	165.0	187.0	114.0	176.0	202.0	137.0	152.0	186.0	136.0	89.0	...	142.0	185.
28	228.0	213.0	154.0	228.0	216.0	156.0	230.0	222.0	160.0	231.0	...	250.0	249.
37	152.0	180.0	121.0	149.0	177.0	118.0	148.0	173.0	115.0	148.0	...	54.0	86.
16	131.0	150.0	104.0	117.0	139.0	90.0	119.0	144.0	89.0	128.0	...	129.0	169.

5 rows × 7501 columns

In [5]:
```
test_df.head()
```

Out[5]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:
2	187.0	208.0	133.0	184.0	205.0	130.0	181.0	202.0	127.0	180.0	...	145.0	179.
5	196.0	207.0	131.0	194.0	205.0	129.0	190.0	200.0	127.0	186.0	...	174.0	186.
8	137.0	171.0	87.0	139.0	173.0	89.0	141.0	175.0	91.0	143.0	...	150.0	184.
19	204.0	216.0	142.0	197.0	208.0	130.0	190.0	202.0	120.0	187.0	...	178.0	188.

24	201.0	201.0	129.0	202.0	201.0	134.0	203.0	201.0	140.0	209.0	...	207.0	199.(

5 rows × 7501 columns

Python class that realizes learning and prediction operations

In [6]:
```
class Classifier:
    __classes_centers = None
    __features = None

    def fit(self, df: pd.DataFrame, train_features: list, target: str):
        self.__features = train_features
        self.__classes_centers = df.groupby(by=target).mean()[train_feat

    def predict(self, df: pd.DataFrame):
        predicted_classes = []
        for current_measure_index in df.index:
            current_measure = df.loc[current_measure_index, self.__featu
            current_predicted_class = self.__get_nearest_class(current_m
            predicted_classes.append(current_predicted_class)
        return predicted_classes

    def __get_nearest_class(self, measure):
        best_class_index = self.__get_best_class_index(measure)
        return self.__classes_centers.index[best_class_index]

    def __get_best_class_index(self, measure):
        tilled_measure = np.tile(measure, (len(self.__classes_centers.in
        difference = self.__classes_centers.values - tilled_measure
        squares = np.square(difference)
        sums = np.sum(squares, axis=1)
        return np.argmin(sums)
```

Train the model and predict classes for test data

In [7]:
```
model = Classifier()
model.fit(train_df, features, "class")
predicted_classes = model.predict(test_df)
```

Calc the accuracy score

In [8]:
```
real_classes = test_df["class"].values
acc = accuracy_score(test_df["class"], predicted_classes)
print(f"The accuracy score is {acc * 100} %")
```

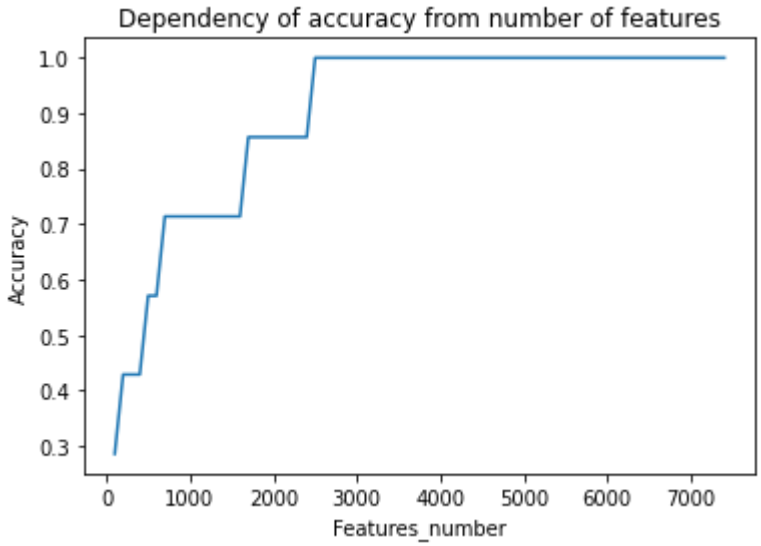
```
The accuracy score is 100.0 %
```

Dependency of accuracy from number of features

In [9]:
```
features_numbers = []
accuracy_values = []
for i in range(100, 7500, 100):
    local_features_to_use = features[:i]
    local_model = Classifier()
    local_model.fit(train_df, local_features_to_use, "class")
    local_predicted_classes = local_model.predict(test_df)
    local_acc = round(accuracy_score(real_classes, local_predicted_class
    features_numbers.append(i)
    accuracy_values.append(local_acc)
```

```
plt.plot(features_numbers, accuracy_values)
plt.title("Dependency of accuracy from number of features")
plt.xlabel("Features_number")
plt.ylabel("Accuracy")
```

Out[9]: Text(0, 0.5, 'Accuracy')



Dependency of accuracy from number of classes

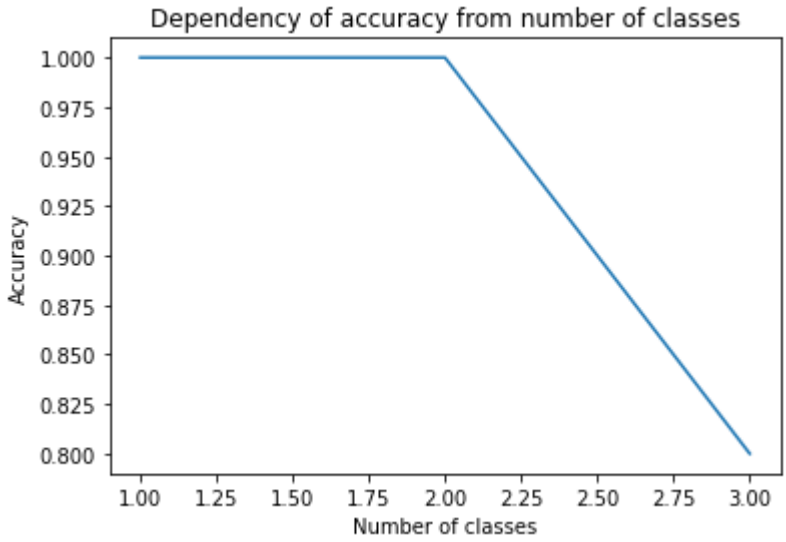
In [10]:

```
class_numbers = []
accuracy_values = []

for class_number in range(1, len(classes), 1):
    local_classes = classes[:class_number]
    local_dataset = dataset[dataset["class"].isin(local_classes)]
    local_train_df = local_dataset.sample(frac=0.85, random_state=18)
    local_test_df = local_dataset.drop(local_train_df.index)
    classes_local_model = Classifier()
    classes_local_model.fit(local_train_df, features, "class")
    local_predicted = classes_local_model.predict(local_test_df)
    real = local_test_df["class"].values
    acc_val = accuracy_score(real, local_predicted)
    class_numbers.append(class_number)
    accuracy_values.append(acc_val)

plt.plot(class_numbers, accuracy_values)
plt.title("Dependency of accuracy from number of classes")
plt.xlabel("Number of classes")
plt.ylabel("Accuracy")
```

Out[10]: Text(0, 0.5, 'Accuracy')



Demonstrate some images

In [11]:
```
Image(filename=f"{data_dir}/water.jpg")
```

Out[11]:

In [12]:
```
Image(filename=f"{data_dir}/town.jpg")
```

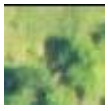
Out[12]:

In [13]:
```
Image(filename=f"{data_dir}/road.jpg")
```

Out[13]:

In [14]:
```
Image(filename=f"{data_dir}/field.jpg")
```

Out[14]:



Висновок

У даному завдання були взяті чотири різні види зображення(вода, місто, поле та ліс). Ці дані були перетворені на вектори фіч та розподілені на вибірки для тренування та навчання. Для навчання використовувався метод еталонів. Модель вказаного программного класу була навчена методом `fit`. Для класифікації був використаний метод `predict` із вказаного программного класу. Модель показала точність 100%. Далі були проведені тести стосовно побудови моделей, що навчаються за різною кількістю ознак. Як видно із малюнку графіку точність у вказаному тесті коливається від 30 до 100 відсотків. Зі збільшенням кількості ознак збільшувалася і точність

розпізнавання. Також був проведений тест із різною кількістю класів розпізнавання і за малюнком графіку було визнано, що зі зменшенням кількості класів точність класифікації підвищується до 100%.

Завдання 2

Постановка задачі

1. Реалізувати детерміноване вирішальне правило на основі методу еталонів, що дробляться. При цьому використати п’яти-рівневі вирішальні правила.

2. За допомогою побудованої правила розв’язати задачу розпізнавання двох стаціонарних за яскравістю зображень. Для цього на основі відповідних графічних файлів необхідно сформувати навчальну та тестову вибірки. Визначити параметри вирішального правила на навчальній вибірці.

3. Перевірити роботу класифікатора на тестових даних.

4. Результати роботи оформити звітом, який має містити: постановку задачі, навчальну вибірку даних у графічному вигляді, результати роботи на тестовій множині даних, всі параметри відтвореного класифікатора, вихідний код.

5. Визначити залежність точності вирішальних правил, сформованих за методом еталонів, що дробляться, від кількості ознак розпізнавання та кількості класів розпізнавання.

6. Визначити ефективність використання методу побудови еталонів як додаткового методу визначення належності реалізацій.

Короткі теоретичні відомості про метод еталонів, які дробляться

Процес навчання за методом еталонів, що дробляться, складається з двох етапів. На першому етапі за навчальною вибіркою для кожного класу будується контейнер у вигляді гіперсфери якомога меншого радіуса, до якої входять всі реалізації класу. При цьому формується еталон кожного класу. обчислюється відстань від еталона до всіх реалізацій даного класу. На другому етапі обирається максимальна з цих відстаней r_{max} . Відтворюється гіперсфера з центром у еталоні і радіусом $R=r_{max}+\varepsilon$. Вона охоплює всі реалізації даного класу. Така процедура проводиться для всіх класів (образів). На рис. 2 наведено приклад двох образів у двовимірному просторі ознак.

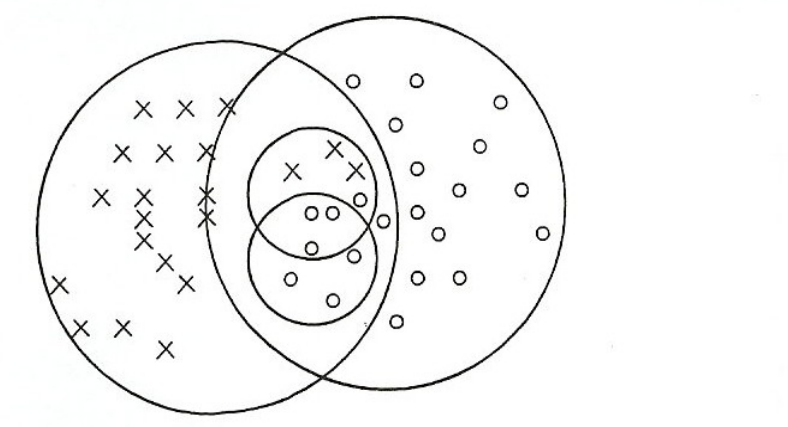


Рисунок 3– Вирішальне правило типу “Метод еталонів, що дробляться”

Якщо гіперсфери різних образів перетинаються і у області перетину містяться реалізації більш ніж одного класу, то для них будуються гіперсфери другого рівня, потім третього і т.д. до тих пір, поки гіперсфери будуть не перетинатися або у області перетину будуть міститися реалізації лише одного класу.

Розпізнавання проводиться таким чином. Визначається місцезнаходження реалізації, належність якої необхідно встановити, відносно гіперсфер першого рівня. Якщо така реалізація розміщується в частині гіперсфери, що відповідає одному и тільки одному класу, то процедура розпізнавання зупиняється. Якщо реалізація розміщується в області перетину гіперсфер, яка при навчанні містить реалізації більш ніж одного класу, то переходимо до гіперсфер другого рівня і проводимо процедуру розпізнавання аналогічну до гіперсфер першого рівня. Такий процес продовжується, поки належність реалізації до певного класу не буде встановлено однозначно. Хоча, така подія може і не настати. Зокрема, невідома реалізація може не потрапити ні в одну з гіперсфер будь-якого рівня. У таких випадках виникає необхідність включити в вирішальне правило відповідні дії Наприклад, система може приймати гіпотезу про неналежність реалізації до жодного з класів, або визначати належність за іншим методом, наприклад, побудови еталонів.

Код реалізації та результати виконання до завдання 2

Import libraries

In [11]:
```
import numpy as np
from math import sqrt
import pandas as pd
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from IPython.display import Image

from labs.helpers import read_dataset
```

Prepare to load data

In [2]:
```
data_dir = "../../data/train"
classes = ["field", "road", "town", "water"]
file_pattern = "*{}*.jpg"
standard_shape = (50, 50, 3)
```

Load data and split it to train and test datasets

In [3]:
```
dataset, features = read_dataset(data_dir, classes, file_pattern, standa
train_df = dataset.sample(frac=0.85, random_state=18)
test_df = dataset.drop(train_df.index)
```

Demonstrate datasets

In [4]:
```
train_df.head()
```

Out[4]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:
36	36.0	72.0	72.0	39.0	75.0	75.0	42.0	78.0	78.0	44.0	...	39.0	74.
26	165.0	187.0	114.0	176.0	202.0	137.0	152.0	186.0	136.0	89.0	...	142.0	185.
28	228.0	213.0	154.0	228.0	216.0	156.0	230.0	222.0	160.0	231.0	...	250.0	249.
37	152.0	180.0	121.0	149.0	177.0	118.0	148.0	173.0	115.0	148.0	...	54.0	86.
16	131.0	150.0	104.0	117.0	139.0	90.0	119.0	144.0	89.0	128.0	...	129.0	169.

5 rows × 7501 columns

In [5]:
```
test_df.head()
```

Out[5]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:
2	187.0	208.0	133.0	184.0	205.0	130.0	181.0	202.0	127.0	180.0	...	145.0	179.
5	196.0	207.0	131.0	194.0	205.0	129.0	190.0	200.0	127.0	186.0	...	174.0	186.
8	137.0	171.0	87.0	139.0	173.0	89.0	141.0	175.0	91.0	143.0	...	150.0	184.

19	204.0	216.0	142.0	197.0	208.0	130.0	190.0	202.0	120.0	187.0	...	178.0	188.
24	201.0	201.0	129.0	202.0	201.0	134.0	203.0	201.0	140.0	209.0	...	207.0	199.

5 rows × 7501 columns

Python class that realizes learning and prediction operations

In [6]:

```
class Classifier:

    head = None
    __features = None
    __classes_centers = None
    __radius = None
    __next_level_predictor = None

    def __init__(self, head=True):
        self.head = head

    def fit(self, df, train_features, class_target, depth=5):
        self.__features = train_features
        self.__classes_centers = df.groupby(by=class_target).mean()[trai
        self.__radius = self.__calc_classes_radius(df, class_target)
        self.__next_level_predictor = self.__build_next_level_predict_ac

    def __calc_classes_radius(self, df, class_target):
        radius = {}
        for class_name in self.__classes_centers.index:
            distances = []
            class_center = self.__classes_centers.loc[class_name].values
            class_df = df[df[class_target] == class_name]
            for measure_idx in class_df.index:
                measure = class_df.loc[measure_idx, self.__features].val
                measure_evclid_distance = self.__calc_evclid_distance(cl
                distances.append(measure_evclid_distance)
            radius[class_name] = max(distances)
        return pd.DataFrame.from_dict(radius, orient="index", columns=["

    def __build_next_level_predict_action(self, df, class_target, depth)
        uncertain_measures = self.__define_uncertain_measures(df)
        if depth != 0 and len(uncertain_measures) != 0:
            next_level_model = Classifier(head=False)
            next_level_model.fit(uncertain_measures, self.__features, cl
            return next_level_model.predict_class

    def __define_uncertain_measures(self, df):
        uncertain_measures_idxs = list()
        for measure_idx in df.index:
            measure = df.loc[measure_idx, self.__features].values
            measure_classes = self.__measure_belong_to_classes(measure)
            if len(measure_classes) > 1:
                uncertain_measures_idxs.append(measure_idx)
        return df.loc[uncertain_measures_idxs]

    def __measure_belong_to_classes(self, measure):
        owner_classes = list()
        for class_name in self.__classes_centers.index:
            class_center = self.__classes_centers.loc[class_name, self.__
            evclid_dist = self.__calc_evclid_distance(measure, class_cen
            class_radius = self.__radius.loc[class_name, "Radius"]
            if evclid_dist < class_radius:
                owner_classes.append(class_name)
```

```
        return owner_classes

    @staticmethod
    def __calc_evclid_distance(measure1, measure2):
        difference = measure1 - measure2
        squared_diff = np.square(difference)
        sum_of_squared_diff = np.sum(squared_diff)
        return sqrt(sum_of_squared_diff)

    def predict(self, df):

        def iterate_test_df(measure_df):
            measure = measure_df.values
            return self.predict_class(measure)

        result = df[self.__features].apply(iterate_test_df, axis=1)
        return result

    def predict_class(self, measure):
        defined_classes = self.__measure_belong_to_classes(measure)
        defined_clases_number = len(defined_classes)
        if defined_clases_number == 1:
            return defined_classes[0]
        elif not self.__next_level_predictor is None:
            defined_class = self.__next_level_predictor(measure)
            if defined_class is None and self.head:
                return self.__predict_class_by_standard_method(measure)
            else:
                return defined_class
        elif not self.head:
            return None
        else:
            return self.__predict_class_by_standard_method(measure)

    def __predict_class_by_standard_method(self, measure):
        distances_to_classes = []
        for class_name in self.__classes_centers.index:
            class_standard = self.__classes_centers.loc[class_name].valu
            evclid_dst = self.__calc_evclid_distance(measure, class_stan
            distances_to_classes.append(evclid_dst)
        best_class_index = np.argmin(distances_to_classes)
        return self.__classes_centers.index[best_class_index]
```

Train the model and predict classes for test data

In [7]:
```
model = Classifier()
model.fit(train_df, features, "class")
pred_y = model.predict(test_df)
```

In [8]:
```
true_y = test_df["class"]
accuracy_score(true_y, pred_y)
```

Out[8]: 1.0

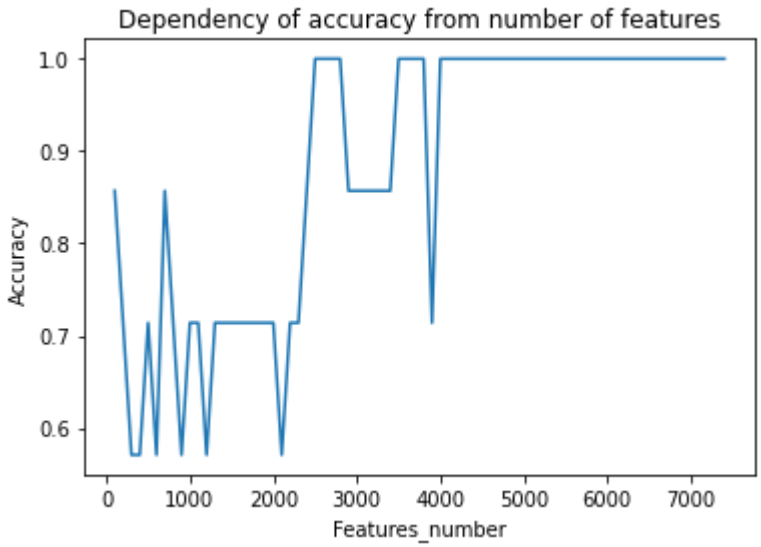
Dependency of accuracy from number of features

In [9]:
```
features_numbers = []
accuracy_values = []
for i in range(100, 7500, 100):
    local_features_to_use = features[:i]
```

```
    local_model = Classifier()
    local_model.fit(train_df, local_features_to_use, "class")
    local_predicted_classes = local_model.predict(test_df)
    local_acc = round(accuracy_score(true_y, local_predicted_classes), 3
    features_numbers.append(i)
    accuracy_values.append(local_acc)

plt.plot(features_numbers, accuracy_values)
plt.title("Dependency of accuracy from number of features")
plt.xlabel("Features_number")
plt.ylabel("Accuracy")
```

Out[9]: Text(0, 0.5, 'Accuracy')



Dependency of accuracy from number of classes

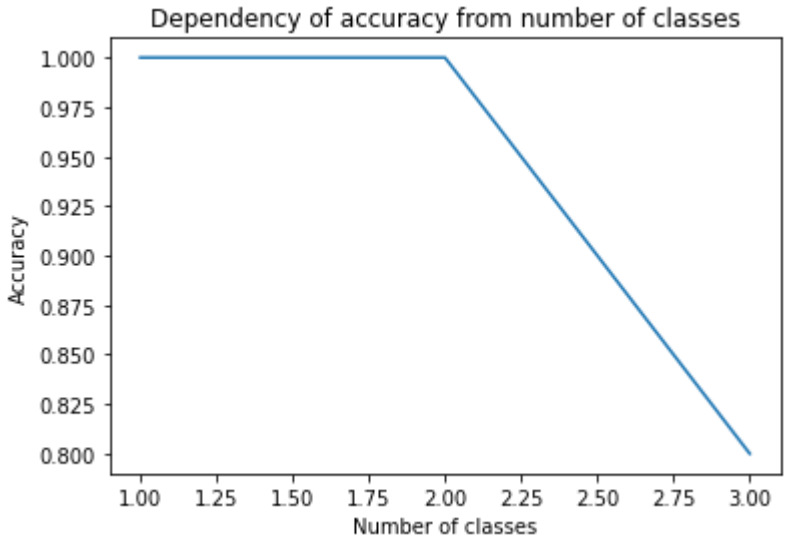
In [10]:

```
class_numbers = []
accuracy_values = []

for class_number in range(1, len(classes), 1):
    local_classes = classes[:class_number]
    local_dataset = dataset[dataset["class"].isin(local_classes)]
    local_train_df = local_dataset.sample(frac=0.85, random_state=18)
    local_test_df = local_dataset.drop(local_train_df.index)
    classes_local_model = Classifier()
    classes_local_model.fit(local_train_df, features, "class")
    local_predicted = classes_local_model.predict(local_test_df)
    real = local_test_df["class"].values
    acc_val = accuracy_score(real, local_predicted)
    class_numbers.append(class_number)
    accuracy_values.append(acc_val)

plt.plot(class_numbers, accuracy_values)
plt.title("Dependency of accuracy from number of classes")
plt.xlabel("Number of classes")
plt.ylabel("Accuracy")
```

Out[10]: Text(0, 0.5, 'Accuracy')



Demonstrate some images

In [12]:
```
Image(filename=f"{data_dir}/water.jpg")
```

Out[12]:

In [13]:
```
Image(filename=f"{data_dir}/town.jpg")
```

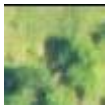
Out[13]:

In [14]:
```
Image(filename=f"{data_dir}/road.jpg")
```

Out[14]:

In [15]:
```
Image(filename=f"{data_dir}/field.jpg")
```

Out[15]:



Висновок

У даному завданні були взяті чотири різні види зображення(вода, місто, поле та ліс). Ці дані були перетворені на вектори фіч та розподілені на вибірки для тренування та навчання. Для навчання використовувався метод еталонів, що дробляться із максимальною кількістю дроблень в 5 разів. Модель вказаного программного класу була навчена методом `fit`. Для класифікації був використаний метод `predict` із вказаного программного класу. Модель показала точність 100%. Далі були проведені тести стосовно побудови моделей, що навчаються за різною кількістю ознак. Як видно із малюнку графіку точність у вказаному тесті коливається від 25 до 100

відсотків. Зі збільшенням кількості ознак збільшувалася і точність розпізнавання. Також був проведений тест із різною кількістю класів розпізнавання і за малюнком графіку було визнано, що зі зменшенням кількості класів точність класифікації підвищується до 100%.

Завдання 3

Постановка задачі

1. Реалізувати одношаровий персептрон.

2. За допомогою реалізованого персептрона розв’язати задачу. Для цього необхідно випадковим чином сформувати навчальну та тестову вибірки (у співвідношенні 4:1). Навчити нейронну мережу на навчальній вибірці, використовуючи алгоритм Розенблатта.

3. Перевірити роботу персептрона на тестових даних.

4. Результати роботи оформити звітом, який має містити: постановку задачі, навчальну вибірку даних та їх представлення у графічному виді на R^2, результати роботи на тестовій множині даних, параметри персептрона, що навчився, вихідний код програми.

Короткі теоретичні відомості про модель персептрона

Модель персептрона має вигляд, показаний на Рисунок 9.

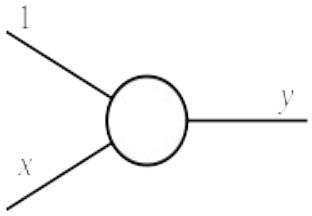


Рисунок 9 – Модель персептрона

При цьому

$$x \in R^d, \text{ або } x \in \{-1, 1\}^d,$$

$$y \in R, \text{ або } y \in \{-1, 1\}.$$

Таку модель використовують для розв’язання задачі класифікації для двох класів і є ідентичною до задачі

$$y \in \{0, 1\}.$$

Будемо розглядати випадок

$$x \in R^d, y \in \{-1, 1\}.$$

Функціонування персептрона описується наступною залежністю:

$$y = \text{sign}(W^T x - \boldsymbol{\tau}) = f(x, W). \quad (3.1)$$

де $\boldsymbol{\tau}$ – деякий поріг, W – вектор вагових коефіцієнтів персептрона.

У геометричній інтерпретації рівняння (3.1) визначає два підпростори

$$\begin{gathered}\{x : y = 1\} \quad \Leftrightarrow \quad H^{+} = \{x : W^{T}x \quad ¿ \quad \boldsymbol{\tau}\}, \\ \{x : y = -1\} \quad \Leftrightarrow \quad H^{-} = \{x : W^{T}x < \boldsymbol{\tau}\},\end{gathered} \tag{3.2}$$

з роздільною гіперплощиною (афінний підпростір розмірності $d - 1$):

$$H = \{x : W^{T}x - \boldsymbol{\tau} = 0\}. \tag{3.3}$$

Збільшуючи розмірність простору, отримаємо

$$x \quad \in \quad R^{d} \quad \Rightarrow \quad \tilde{x} \quad \in \quad R^{d+1}, \tag{3.4}$$

де $\tilde{x}_i = x_i$, $\tilde{x}_{d+1i} = 1$, i ¿ d,

$$W \quad \in \quad R^{d} \quad \Rightarrow \quad \tilde{W} \quad \in \quad R^{d+1}, \tag{3.5}$$

де $\tilde{W}_i = W_i$, $\tilde{W}_{d+1i} = \boldsymbol{\tau}$, i ¿ d.

Враховуючи (3.4) та (3.5), можна записати

$$W^{T}x - \boldsymbol{\tau} = \tilde{W}^{T}\tilde{x}.$$

Навчання персептрона (алгоритм Розенблатта)

Навчання персептрона представляє собою процес налаштування вагових коефіцієнтів W. При навчанні нейронної мережі, як правило, математичні вирази для розділяючих поверхонь відсутні. Тому навчання виконується тільки на навчальній вибірці.

Навчальна вибірка (скінчена) задається множиною, що складається з пар вхід-вихід:

$$T = \{(x_1, t_1), \ldots, (x_n, t_n)\} = \{(x_i, t_i), i = 1, \ldots, n\}, \tag{3.6}$$

де $t_i \in \{-1, 1\}$.

Мета навчання – налаштувати вагові коефіцієнти W таким чином, щоб для будь-яких виконувалось $x^* \in R^d$, $x^* \notin T$ виконувалось $y = t^*$.

Алгоритм навчання персептрона Розенблатта:

Даний алгоритм коректно працює лише в тих випадках, коли класи є лінійно роздільними.

1. Формуємо множину

$$\tilde{F} = \tilde{S}^{+} \cup \left(-\tilde{S}^{-}\right) \subset R^{d+1},$$

де

$$\tilde{S}^{+} = \{x: \text{якщо існує } i \text{ таке, що } t_i = 1, x = x_i\},$$
$$\tilde{S}^{-} = \{x: \text{якщо існує } i \text{ таке, що } t_i = -1, x = x_i\}$$
$$-\tilde{S}^{-} = \{\tilde{z} : \forall \tilde{x}_i \in \tilde{S}^{-}, \tilde{z}_i = t_i \tilde{x}_i\},$$

і систему

$$\tilde{W}^{T} \tilde{z} > 0 \text{ для будь яких } \tilde{z} \in \tilde{F}.$$

2. Початок. Вибираємо деякий елемент $\tilde{z} \in \tilde{F}$ як початкове наближення для $\tilde{W}$. Сформуємо випадкову послідовність (циклічну, у якій елементи з'являються з невизначеною частотою) з елементів $\tilde{W}$.

3. Тест. Вибираємо випадкове значення $\tilde{z}_{ij} = \text{rand}(\tilde{F})$. Якщо $\tilde{W}^{T} \tilde{z}_{ij} > 0$, переходимо до п. 3, інакше — до п. 4.

4. Модифікація вагових коефіцієнтів.

$$\tilde{w} = \tilde{w} + \phi \tilde{z}_{ij},$$

де $\phi = 1$.

(Операції 4 обумовлені пошуком розв’язку $\tilde{W}$ у формі

$$\tilde{W} = \sum_j \alpha_j \tilde{z}_j, \quad \alpha_j > 0.$$

Крім того

$$\tilde{W}_j \cdot \tilde{z}_j = \tilde{W}_{j-1} \cdot \tilde{z}_{ij} + \phi > \tilde{w}_{j-1} \cdot \tilde{z}_{ij}.$$

Значення $\tilde{W}_j$ – збільшується, щоб після поточного негативного значення на наступному кроці було отримане додатне (п. 4 виконується тільки у випадку негативного добутку). Переходимо до п. 3.

5. Завершення. Процес навчання закінчується тоді, коли умова $\tilde{W}^{T} \tilde{z}_{ij} > 0$ буде виконуватися для всіх векторів навчальної вибірки.

Код реалізації та результати роботи до завдання 3

Import libraries

In [13]:
```
from scipy.special import expit
import numpy as np
from sklearn.metrics import accuracy_score
from IPython.display import Image

from labs.helpers import read_dataset
```

Prepare to load data

In [14]:
```
data_dir = "../../data/train"
classes = ["field", "water"]
file_pattern = "*{}*.jpg"
standard_shape = (50, 50, 3)
```

Load data and split it to train and test datasets

In [15]:
```
dataset, features = read_dataset(data_dir, classes, file_pattern, standa
train_df = dataset.sample(frac=0.8, random_state=18)
test_df = dataset.drop(train_df.index)
```

Demonstrate datasets

In [16]:
```
train_df.head()
```

Out[16]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:2
0	0.0	7.0	3.0	11.0	24.0	15.0	0.0	10.0	0.0	10.0	...	156.0	199.0
13	48.0	83.0	76.0	51.0	86.0	79.0	55.0	90.0	83.0	54.0	...	52.0	85.0
18	58.0	92.0	91.0	61.0	95.0	94.0	67.0	101.0	100.0	73.0	...	73.0	103.0
12	61.0	87.0	74.0	70.0	96.0	83.0	80.0	107.0	90.0	94.0	...	97.0	129.0
7	126.0	168.0	102.0	125.0	168.0	99.0	123.0	166.0	97.0	123.0	...	175.0	196.0

5 rows × 7501 columns

In [17]:
```
test_df.head()
```

Out[17]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:
5	196.0	207.0	131.0	194.0	205.0	129.0	190.0	200.0	127.0	186.0	...	174.0	186.0
10	212.0	217.0	161.0	214.0	219.0	163.0	212.0	220.0	163.0	204.0	...	146.0	190.0
17	66.0	105.0	86.0	64.0	103.0	84.0	62.0	101.0	82.0	59.0	...	153.0	173.0
19	55.0	87.0	82.0	54.0	86.0	81.0	53.0	85.0	80.0	53.0	...	178.0	196.0

4 rows × 7501 columns

Python class that realizes learning and prediction operations

In [18]:

```
class LinearPerceptron:

    __features = None
    __class_to_number = None
    __number_to_class = None
    weights = None
    __old_weights_delta = None

    def __init__(self, learning_rate=5, moment=0.7, max_epoch=100000):
        self.__learning_rate = learning_rate
        self.__moment = moment
        self.__max_epoch = max_epoch
        self.__activation = expit

    def fit(self, df, train_features, target):
        self.__features = train_features
        self.__build_classes_binarizers(df, target)
        train_values = self.__build_values(df)
        binarized_classes = df[target].apply(lambda class_: self.__class_
        self.__old_weights_delta = np.zeros(train_values.shape[1])
        self.weights = np.full(train_values.shape[1], 0.001)
        # inside for operator
        for epoch_number in range(self.__max_epoch):
            cum_error = 0
            for measure_index in range(train_values.shape[0]):
                measure = train_values[measure_index]
                real_value = binarized_classes[measure_index]
                cum_error += self.__learn(measure, real_value)
                mean_epoch_error = np.mean(cum_error)
                if mean_epoch_error < 0.000000001:
                    return

    def __build_values(self, df):
        measures = df[self.__features].values
        bias_neuron_vals = np.ones((measures.shape[0], 1))
        values = np.hstack((bias_neuron_vals, self.__activation(measures
        return values

    def __learn(self, measure, real_value):
        neuron_output = self.__go_forward(measure)
        self.__go_backward(measure, neuron_output, real_value)
        return self.__go_forward(measure)

    def __go_forward(self, measure):
        multiplied = np.multiply(measure, self.weights)
        neuron_input = np.sum(multiplied)
        neuron_output = self.__activation(neuron_input)
        return neuron_output

    def __go_backward(self, measure, neuron_output, real_value):
        out_neuron_error = (real_value - neuron_output) * self.__activat
        neuron_activation_derivation = np.multiply(1 - measure, measure)
        error_by_weights = out_neuron_error * self.weights
        in_neurons_errors = np.multiply(neuron_activation_derivation, er
        gradients = np.multiply(in_neurons_errors, measure)
        new_weights_delta = self.__learning_rate * gradients + self.__ol
        self.weights += new_weights_delta
        self.__old_weights_delta = new_weights_delta
```

```
    def __build_classes_binarizers(self, df, target):
        self.__class_to_number = dict()
        self.__number_to_class = dict()
        classes_list = df[target].unique()
        for k, v in enumerate(classes_list[:2]):
            self.__class_to_number[v] = k
            self.__number_to_class[k] = v

    @staticmethod
    def __activation_derivative(neuron_out):
        return (1 - neuron_out) * neuron_out

    def predict(self, df):

        def iterate_over_measures(measure):
            result = self.__go_forward(measure)
            return self.__number_to_class[int(round(result))]

        test_values = self.__build_values(df)
        return np.apply_along_axis(iterate_over_measures, 1, test_values
```

Train the model and predict classes for test data

In [19]:
```
model = LinearPerceptron()
model.fit(train_df, features, "class")
print("Model trained with such weights:")
model.weights
```

```
Model trained with such weights:
```

Out[19]:
```
array([1.00000000e-003, 6.03510639e-135, 2.03087948e-007, ...,
       1.00000000e-003, 1.00000000e-003, 1.00000000e-003])
```

In [20]:
```
real_classes = test_df["class"].values
predicted = model.predict(test_df)

print(f"Accuracy is {accuracy_score(real_classes, predicted)}")
```

```
Accuracy is 0.5
```

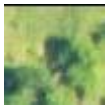
Demonstrate some images

In [25]:
```
Image(filename=f"{data_dir}/water.jpg")
```

Out[25]:

In [26]:
```
Image(filename=f"{data_dir}/field.jpg")
```

Out[26]:



Висновок

У даному завданні були взяті два різні види зображення(поле та вода). Ці дані були перетворені на вектори фіч та розподілені на вибірки для тренування та навчання. Для навчання використовувався лінійний одношаровий персептрон. Модель вказаного программного класу була навчена методом `fit`. Для класифікації був використаний метод `predict` із вказаного программного класу. Модель показала точність 50%. Можливими проблемами такої низької якості є: недостатня кількість епох навчання, не пудубрані гіперпараметри швидкості навчання та моменту, мала кількість реалізацій.

Завдання 4

Постановка задачі

1. Реалізувати метод потенційних функцій для варіанту, коли потенційна функція спадає в значній мірі в залежності від відстані та коли вона є менш крутою.
2. За допомогою реалізованого класифікатора розв’язати задачу. Для цього необхідно випадковим чином сформувати навчальну та тестову вибірки (у співвідношенні 4:1).
3. Перевірити роботу методу потенційних функцій на тестових даних.
4. Результати роботи оформити звітом, який має містити: постановку задачі, навчальну вибірку даних та їх представлення у графічному виді на R^2, результати роботи на тестовій множині даних, параметри вирішального правила, вихідний код програми.

Короткі теоретичні відомості про метод потенційних функцій

Назва метода в певній мірі пов’язана з такою аналогією. Будемо вважати, що розпізнається два образа. Уявімо, що об’єкти є точками $\overline{x}_j$, деякого простору $\boldsymbol{X}$. У ці точки будемо розміщувати заряди $+\mathbf{q_j}$, якщо об’єкт належить образу $\boldsymbol{S_1}$, і $-\mathbf{q_j}$, якщо об’єкт належить образу $\boldsymbol{S_2}$ (рис. 11).

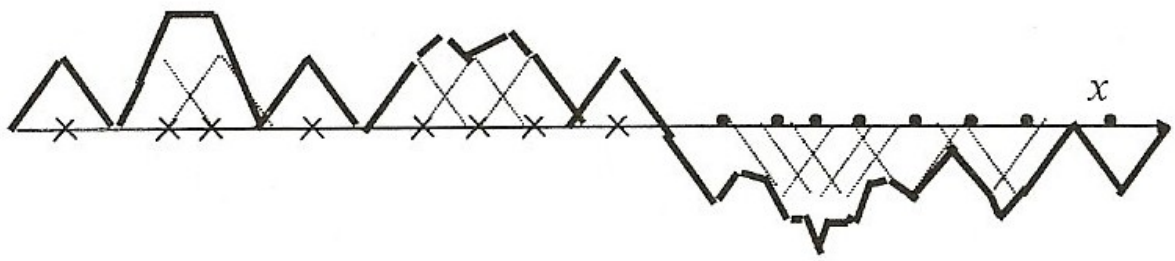


Рисунок 11 – Синтез потенційної функції в процесі навчання

Тут
——— – потенційна функція, що формується одиночним об’єктом;
▬▬▬ – сумарна потенційна функція, що формується навчальною послідовністю.

Функцію, що описує розподіл електростатичного потенціалу в такому полі, можна використовувати як вирішальне правило (або для його побудови). Якщо потенціал точки $\overline{x}$, що створюється одиничним зарядом, який знаходиться у $\overline{x}_j$, дорівнює $K(\overline{x}, \overline{x}_j)$, то загальний потенціал в $\overline{x}$, що створюється n зарядами, дорівнює

$$g(\overline{x}) = \sum_{j=1}^{n} q_j K(\overline{x}, \overline{x_j}),$$

де $K(\bar{x}, \bar{x}_j)$ – потенційна функція. Вона, як і у фізиці, спадає при збільшенні евклідової відстані між $\bar{x}$ та $\bar{x}_j$. Часто як потенційна використовується функція, що має максимум при $\bar{x} = \bar{x}_j$ і монотонно спадає до нуля при $\|\bar{x} - \bar{x}_j\| \to \infty$.

Розпізнавання може виконуватися в такий спосіб. В точці $\bar{x}$, де знаходиться об’єкт, що розпізнається, обчислюється потенціал $g(\bar{x})$. Якщо він виявляється додатнім, то об’єкт відносять до образу S_1. Якщо від’ємним – до образу S_2.

При значному обсязі навчальної вибірки ці обчислення є достатньо громіздкими, і ефективніше не обчислювати $g(\bar{x})$, а оцінювати роздільну межу класів (образів), або апроксимувати потенційне поле.

Код реалізації та результати роботи до завдання 4

Import libraries

In [46]:
```
from math import sqrt, pow
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from IPython.display import Image
from functools import partial

from labs.helpers import read_dataset
```

Prepare to load data

In [47]:
```
data_dir = "../../data/train"
classes = ["field", "water"]
file_pattern = "*{}*.jpg"
standard_shape = (50, 50, 3)
```

Load data and split it to train and test datasets

In [48]:
```
dataset, features = read_dataset(data_dir, classes, file_pattern, standa
train_df = dataset.sample(frac=0.8, random_state=18)
test_df = dataset.drop(train_df.index)
```

Demonstrate datasets

In [49]:
```
train_df.head()
```

Out[49]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:2
0	0.0	7.0	3.0	11.0	24.0	15.0	0.0	10.0	0.0	10.0	...	156.0	199.0
13	48.0	83.0	76.0	51.0	86.0	79.0	55.0	90.0	83.0	54.0	...	52.0	85.0
18	58.0	92.0	91.0	61.0	95.0	94.0	67.0	101.0	100.0	73.0	...	73.0	103.0
12	61.0	87.0	74.0	70.0	96.0	83.0	80.0	107.0	90.0	94.0	...	97.0	129.0
7	126.0	168.0	102.0	125.0	168.0	99.0	123.0	166.0	97.0	123.0	...	175.0	196.0

5 rows × 7501 columns

In [50]:
```
test_df.head()
```

Out[50]:

	1:1:1	1:1:2	1:1:3	1:2:1	1:2:2	1:2:3	1:3:1	1:3:2	1:3:3	1:4:1	...	50:48:1	50:48:
5	196.0	207.0	131.0	194.0	205.0	129.0	190.0	200.0	127.0	186.0	...	174.0	186.0
10	212.0	217.0	161.0	214.0	219.0	163.0	212.0	220.0	163.0	204.0	...	146.0	190.0
17	66.0	105.0	86.0	64.0	103.0	84.0	62.0	101.0	82.0	59.0	...	153.0	173.0

19	55.0	87.0	82.0	54.0	86.0	81.0	53.0	85.0	80.0	53.0	...	178.0	196.

4 rows × 7501 columns

Python class that realizes learning and prediction operations

In [59]:

```
class Classifier:
    __features = None
    __target = None
    __classes = None
    window_width = 100
    __all_measures = None
    weight_column_name = "weight"

    def fit(self, df, train_features, target):
        self.__features = train_features
        self.__target = target
        self.__classes = df[self.__target].unique()
        self.__all_measures = df[train_features + [target]].copy()
        self.__all_measures[self.weight_column_name] = 0
        while True:
            continue_learning = False
            for index, feature_values in self.__all_measures.iterrows():
                real_class = feature_values[self.__target]
                measure = feature_values[self.__features].values
                best_potential_class = self.__define_best_potential_clas
                if best_potential_class != real_class:
                    continue_learning = True
                    self.__all_measures[self.weight_column_name] += 1
            if not continue_learning:
                break

    def __define_best_potential_class(self, measure):
        potentials = self.__calc_potentials_for_classes(measure)
        if potentials[0] == potentials[1]:
            return ""
        best_potential_class_index = np.argmax(potentials)
        best_potential_class = self.__classes[best_potential_class_index
        return best_potential_class

    def __calc_potentials_for_classes(self, measure):
        classes_potentials = []
        func_to_iterate = partial(self.__calc_weight_function, measure)
        for class_name in self.__classes:
            class_df = self.__all_measures[self.__all_measures[self.__ta
            result = class_df[self.__features + [self.weight_column_name
            classes_potentials.append(sum(result.values))
        return classes_potentials

    def __calc_weight_function(self, measure1, measure_weight_2):
        measure2 = measure_weight_2[self.__features].values
        weight = measure_weight_2[self.weight_column_name]
        dist = self.__calc_evclid_distance(measure1, measure2) / self.wi
        return weight * self.calc_potential_func(dist)

    @staticmethod
    def calc_potential_func(x):
        return 1 / (x + 1)

    @staticmethod
    def __calc_evclid_distance(measure1, measure2):
        difference = measure1 - measure2
```

```
            squared_diff = np.square(difference)
            sum_of_squared_diff = np.sum(squared_diff)
            return sqrt(sum_of_squared_diff)

        @property
        def weights(self):
            return self.__all_measures[self.weight_column_name].values

        def predict(self, df):
            classes_ = df[self.__features].apply(self.__define_best_potentia
            return classes_
```

Train the model and predict classes for test data

In [60]:

```
model = Classifier()
model.fit(train_df, features, "class")
print(f"Model stopped learning with such weights : {model.weights}")
```

```
Model stopped learning with such weights : [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1]
```

In [61]:

```
predicted_classes = model.predict(test_df)
real_classes = test_df["class"].values
acc_sc = accuracy_score(real_classes, predicted_classes)
print(f"Accuracy is {acc_sc * 100} %")
```

```
Accuracy is 100.0 %
```

Dependency of accuracy from number of features

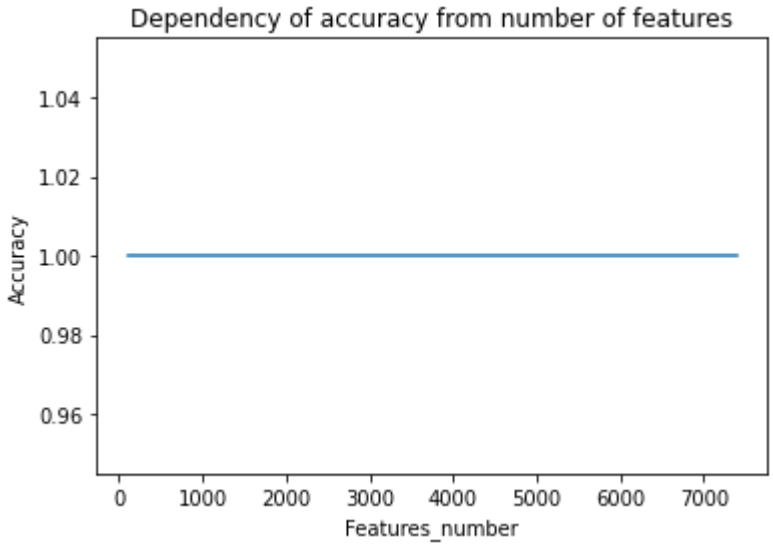
In [54]:

```
features_numbers = []
accuracy_values = []
for i in range(100, 7500, 100):
    local_features_to_use = features[:i]
    local_model = Classifier()
    local_model.fit(train_df, local_features_to_use, "class")
    local_predicted_classes = local_model.predict(test_df)
    local_acc = round(accuracy_score(real_classes, local_predicted_classe
    features_numbers.append(i)
    accuracy_values.append(local_acc)

plt.plot(features_numbers, accuracy_values)
plt.title("Dependency of accuracy from number of features")
plt.xlabel("Features_number")
plt.ylabel("Accuracy")
```

Out[54]:

```
Text(0, 0.5, 'Accuracy')
```



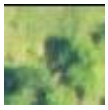
Demonstrate some images

In [55]:
```
Image(filename=f"{data_dir}/water.jpg")
```

Out[55]:

In [58]:
```
Image(filename=f"{data_dir}/field.jpg")
```

Out[58]:



Висновок

У даному завданні були взяті два різні види зображення(вода та поле). Ці дані були перетворені на вектори фіч та розподілені на вибірки для тренування та навчання. Для навчання використовувався метод потенційних функцій. Модель вказаного программного класу була навчена методом `fit`. Для класифікації був використаний метод `predict` із вказаного программного класу. Модель показала точність 100%. Далі були проведені тести стосовно побудови моделей, що навчаються за різною кількістю ознак. Як видно із малюнку графіку точність у вказаному тесті становить 100 відсотків не залежно від кількості ознак. Дана аномалія можлива через невеликий об'єм тренувальних та тестових даних, а також через велику розмірність.

Висновки

В даній роботі було розглянуто 4 методи, які можна застосовувати для задачі розпізнавання образів.

Першим із розглянутих був метод еталонів. Значною перевагою для нього є те, що він простий в реалізації, проте точність методу не висока в тому випадку коли класи перетинаються.

Наступним методом ми розглянули метод еталонів, що дробляться. Це також доволі простий в реалізації ознак. На відміну від попереднього цей метод дозволяє побудувати декілька вирішальних правил про класифікацію класу.

Третій розглянутий метод це застосування персептрону Розенблата в задачі побудови лінійного вирішального правила. Даний метод дозволяє побудувати розділяючу гіперповерхню у вигляді лінійної функції. Персептрон використовується для підбору коефіцієнтів лінійної функції, котра розділяє два класи, які розпізнаються. Нажаль лінійні вирішальні правила не підходять для випадку коли класи розпізнавання перетинаються. У випадку коли класи перетинаються навчальна вибірка не є нормально розподіленою, а вектори-реалізації образів не є лінійно роздільними. Точність розпізнавання в такому випадку доволі низька.

Четвертий метод, який був розглянутий це метод потенційних функцій. Ідея даного методу полягає в підборі деякої функції, яка визначає приналежність об’єкту до одного з класів розпізнавання з деякої похибкою розпізнавання. Основною складністю до застосування даного методу є задача складності підбору підходящої функції для визначення потенціалу.