Отчёт по лабораторной работе №5 "Переобучение и регуляризация"

In [1]:

```
import numpy as np
import pandas as pd
from matplotlib import cm
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from os.path import isfile, join
from os import listdir
import collections
import nltk
import html
import re

DATA_FILE_NAME_1 = 'Lab 5/ex5data1'
DATA_FILE_NAME_2 = 'Lab 5/ex5data2'
DATA_FILE_NAME_3 = 'Lab 5/ex5data3'
DATA_FILE_NAME_4 = 'Lab 5/spamTrain'
DATA_FILE_NAME_5 = 'Lab 5/spamTest'
DATA_FILE_NAME_6 = 'Lab 5/vocab'
```

Imports from 'common' file:

In [2]:

```
import os
from scipy.io import loadmat

DATA_DIRECTORY = '../Data/'
CUSTOM_DATA_DIRECTORY = '../CustomData/'
COLORS = ['r', 'g', 'b', 'gold', 'orange', 'black', 'brown', 'cyan', 'magenta']
MARKERS = ['x', 'o', '.', '+']


def load_data(filename, convert_type=float, separator=',', directory=DATA_DIRECTORY, encoding='utf-
8',
              skip_extention=False, split_function=None):

    filepath = directory + filename
    if not skip_extention:
        filepath += '.txt'

    if not split_function:
        def split_function(line):
            return line.replace('\n', '').split(separator)

    data = []
    with open(filepath, 'r', encoding=encoding) as f:
        for line in f.readlines():
            try:
                if isinstance(convert_type, (list, tuple)):
                    data.append([
                        convert_type[j](el)
                        for row in line.replace('\n', '')
                        for j, el in enumerate(row.split(separator))
                    ])
                else:
                    data.append([convert_type(x) for x in split_function(line)])
            except TypeError:
                pass

    return np.array(data)


def load_data_from_mat_file(filename, directory=DATA_DIRECTORY):
```

```
    filepath = directory + f'{filename}.mat'
    return loadmat(filepath)
```

1. Загрузите данные ex5data1.mat из файла.

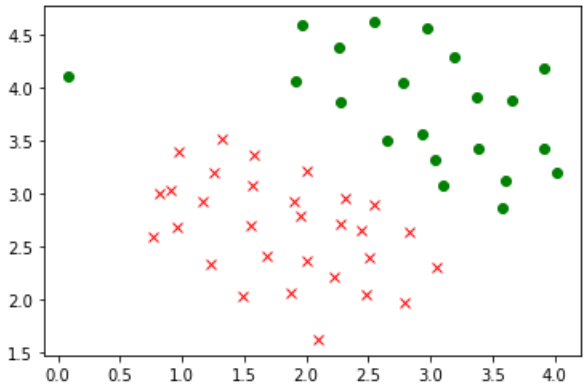
In [3]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_1)
df_1 = pd.DataFrame({'x1': df_data['X'][:, 0], 'x2': df_data['X'][:, 1], 'y': df_data['y'][:, 0]})
X = df_1[['x1', 'x2']]
Y = df_1['y']
```

2. Постройте график для загруженного набора данных: по осям - переменные X^1, X^2, а точки, принадлежащие различным классам должны быть обозначены различными маркерами.

In [4]:

```
def plot_data():
    for i, group in df_1.groupby(['y']):
        plt.plot(group.x1, group.x2, COLORS[i] + MARKERS[i])


plot_data()
plt.show()
```



3. Обучите классификатор с помощью библиотечной реализации SVM с линейным ядром на данном наборе.

In [5]:

```
classifier_c1 = SVC(kernel='linear', C=1.0)
classifier_c1.fit(X, Y)
```

Out[5]:

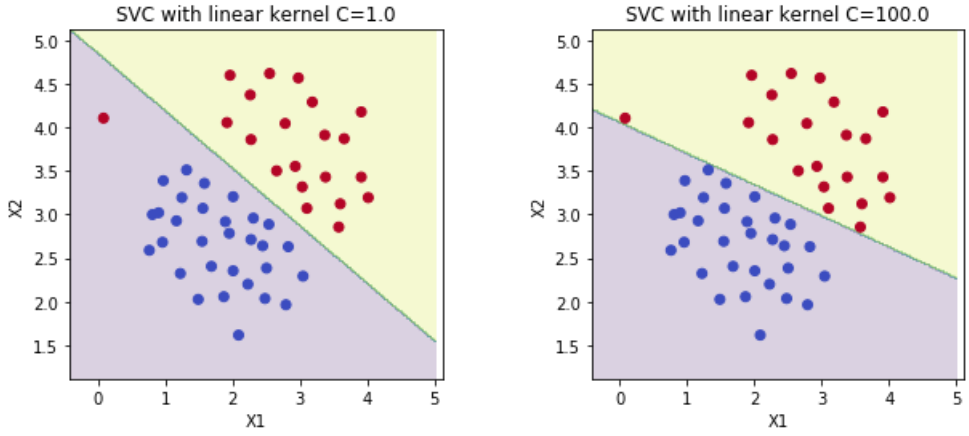
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

4. Постройте разделяющую прямую для классификаторов с различными параметрами C = 1, C = 100 (совместно с графиком из пункта 2). Объясните различия в полученных прямых?

In [6]:

```
def plot_decision_boundary(classifiers_map, X, Y, **kwargs):
    kwargs.setdefault('contour_params', {})
    kwargs.setdefault('scatter_params', {})
    kwargs.setdefault('bias', 0.5)
```

```
    bias = kwargs.get('bias')
    h = .02

    x_min, x_max = X.x1.min() - 1, X.x1.max() + 1
    y_min, y_max = X.x2.min() - 1, X.x2.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = classifiers_map[0][1].predict(np.c_[xx.ravel(), yy.ravel()])
    plt.figure(figsize=(5 * len(classifiers_map), 4))

    for i, (title, classifier) in enumerate(classifiers_map):
        plt.subplot(1, len(classifiers_map), i + 1)
        plt.subplots_adjust(wspace=0.4, hspace=0.4)
        Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, alpha=0.2, **kwargs['contour_params'])

        plt.scatter(X.x1, X.x2, c=Y, cmap=cm.coolwarm, **kwargs['scatter_params'])
        plt.xlabel('X1')
        plt.ylabel('X2')
        plt.xlim(X.x1.min() - bias, X.x2.max() + bias)
        plt.ylim(X.x2.min() - bias, X.x2.max() + bias)
        plt.title(title)

    plt.show()


classifiers_map = [(f'SVC with linear kernel C={c}', SVC(kernel='linear', C=c)) for c in [1., 100.]
]
[c.fit(X, Y) for title, c in classifiers_map]

plot_decision_boundary(classifiers_map, X, Y)
```



Из графиков можно заметить, то при C = 100 разделяющая кривая смещается в сторону отклоняющегося примера. Это происходит потому, что, чем больше коефицент C, тем модель более чувствительна к новым данным (high variance), то есть больше склонна к переобучению. Параметр регуляризации C равный 1 оказался выбран достаточно удачно, так как модель не придаёт большого значения одному отклоняющемуся примеру.

5. Реализуйте функцию вычисления Гауссового ядра для алгоритма SVM.

In [7]:

```
def get_gaussian_kernel_function(sigma=1.):
    def func(X, L):
        F = [np.exp(-((X - l) ** 2).sum(axis=1) / (2 * sigma ** 2)) for l in L]
        return np.array(F).T

    return func


gaussian_kernel_function = get_gaussian_kernel_function()
kernel = gaussian_kernel_function(X.values, X.values)
```

6. Загрузите данные ex5data2.mat из файла.

In [8]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_2)
df_2 = pd.DataFrame({'x1': df_data['X'][:, 0], 'x2': df_data['X'][:, 1], 'y': df_data['y'][:, 0]})
X = df_2[['x1', 'x2']]
Y = df_2['y']
```

7. Обработайте данные с помощью функции Гауссового ядра.

In [9]:

```
gaussian_kernel_function = get_gaussian_kernel_function(0.1)
kernel = gaussian_kernel_function(X.values, X.values)
```

8. Обучите классификатор SVM.

In [10]:

```
gaussian_classifier = SVC(kernel=gaussian_kernel_function, C=1., gamma='scale')
gaussian_classifier.fit(X.values, Y.values)
```

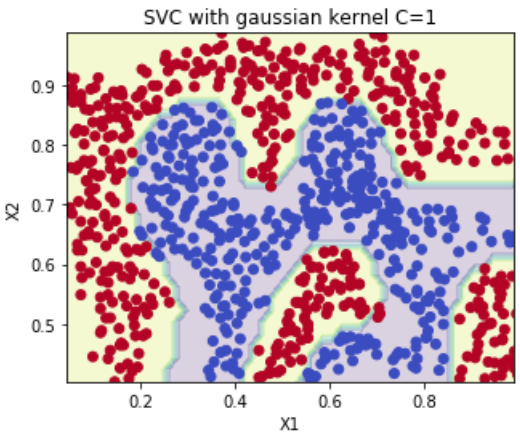
Out[10]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale',
    kernel=<function get_gaussian_kernel_function.<locals>.func at 0x7f559c589200>,
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

9. Визуализируйте данные вместе с разделяющей кривой (аналогично пункту 4).

In [11]:

```
gaussian_classifier_map = ('SVC with gaussian kernel C=1', gaussian_classifier)
plot_decision_boundary([gaussian_classifier_map], X, Y, bias=0.0)
```



10.Загрузите данные ex5data3.mat из файла.

In [12]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_3)
df_3 = pd.DataFrame({'x1': df_data['X'][:, 0], 'x2': df_data['X'][:, 1], 'y': df_data['y'][:, 0]})
X = df_3[['x1', 'x2']]
Y = df_3['y']
df_3val = pd.DataFrame({'x1': df_data['Xval'][:, 0], 'x2': df_data['Xval'][:, 1], 'y': df_data['yva
l'][:, 0]})
Xval = df_3val[['x1', 'x2']]
Yval = df_3val['y']
```

11.Вычислите параметры классификатора SVM на обучающей выборке, а также подберите параметры C и σ^2 на валидационной выборке.

In [13]:

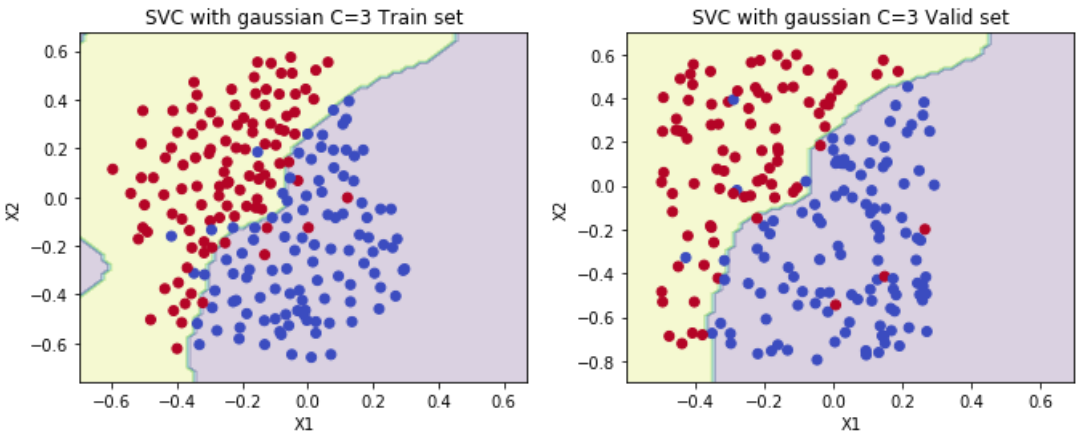
```
def search_optimal_params(X, y, Xval, Yval, C_list, gamma_list):
    best_score = -np.inf
    best_params = None
    for C in C_list:
        for gamma in gamma_list:
            classifier = SVC(kernel='rbf', C=C, gamma=gamma)
            classifier.fit(X, y)
            score = classifier.score(Xval, Yval)
            if score > best_score:
                best_score = score
                best_params = (C, gamma)

    return best_params


values_list = (0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30)
best_params = search_optimal_params(X, Y, Xval, Yval, C_list=values_list, gamma_list=values_list)
C_train, gamma_train = best_params
sigma_train = 1 / (2 * gamma_train)

print(f'Best params for validation set: C = {C_train}, sigma square = {sigma_train}')
```

```
Best params for validation set: C = 3, sigma square = 0.016666666666666666
```

12.Визуализируйте данные вместе с разделяющей кривой (аналогично пункту 4).

In [14]:

```
gaussian_classifiers = [SVC(kernel='rbf', C=C_train, gamma=gamma_train)] * 2
[gaussian_classifiers[i].fit(x.values, y.values) for i, (x, y) in enumerate([(X, Y), (Xval,
Yval)])]

gaussian_classifier_map = (f'SVC with gaussian C={C_train} Train set', gaussian_classifiers[0])
plot_decision_boundary([gaussian_classifier_map], X, Y, bias=0.1)

gaussian_classifier_map = (f'SVC with gaussian C={C_train} Valid set', gaussian_classifiers[1])
plot_decision_boundary([gaussian_classifier_map], Xval, Yval, bias=0.1)
```



13.Загрузите данные spamTrain.mat из файла.

In [15]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_4)
X = np.array(df_data['X'])
Y = np.array(df_data['y'])[:, 0]
```

14.Обучите классификатор SVM.

In [16]:

```
gaussian_classifier = SVC(kernel='rbf', C=1., gamma='scale')
gaussian_classifier.fit(X, Y)
```

Out[16]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

15.Загрузите данные spamTest.mat из файла.

In [17]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_5)
Xtest = np.array(df_data['Xtest'])
Ytest = np.array(df_data['ytest'])[:, 0]
```

16.Подберите параметры C и σ^2.

In [18]:

```
best_params = search_optimal_params(X, Y, Xtest, Ytest,
                                    C_list=np.arange(2, 4, 10), gamma_list=np.linspace(0.0005, 0.00
, 10))
C_train, gamma_train = best_params
sigma_train = 1 / (2 * gamma_train)

print(f'Best params for validation set: C = {C_train}, sigma squared = {sigma_train}', '\n')
```

```
Best params for validation set: C = 2, sigma squared = 100.0
```

17.Реализуйте функцию предобработки текста письма:

Функция должна включать в себя:

- перевод в нижний регистр;
- удаление HTML тэгов;
- замена URL на одно слово (например, “httpaddr”);
- замена email-адресов на одно слово (например, “emailaddr”);
- замена чисел на одно слово (например, “number”);
- замена знаков доллара ($) на слово “dollar”;
- замена форм слов на исходное слово (например, слова “discount”, “discounts”, “discounted”, “discounting” должны быть заменены на слово “discount”). Такой подход называется stemming;
- остальные символы должны быть удалены и заменены на пробелы, т.е. в результате получится текст, состоящий из слов, разделенных пробелами.

In [19]:

```
def prepare_body(body):
    # a
    body = body.lower()

    # b
    text = html.unescape(body)
    body = re.sub(r'<[^>]+?>', '', text)

    # c
    regx = re.compile(r"(http|https)://[^\s]*")
```

```
    regx = re.compile(r"(http|https)://[^\s]*")
    body = regx.sub(repl=" httpaddr ", string=body)

    # d
    regx = re.compile(r"\b[^\s]+@[^\s]+[.][^\s]+\b")
    body = regx.sub(repl=" emailaddr ", string=body)

    # e
    regx = re.compile(r"\b(\d+|\d+\.\d+)\b")
    body = regx.sub(repl=" number ", string=body)

    # f
    regx = re.compile(r"[$]")
    body = regx.sub(repl=" dollar ", string=body)

    # h
    regx = re.compile(r"([^\w\s]+)|([_-]+)")
    body = regx.sub(repl=" ", string=body)
    regx = re.compile(r"\s+")
    body = regx.sub(repl=" ", string=body)

    # g
    body = body.strip(" ")
    bodywords = body.split(" ")
    keepwords = [word for word in bodywords if word not in stopwords.words('english')]
    stemmer = SnowballStemmer("english")
    stemwords = [stemmer.stem(wd) for wd in keepwords]
    body = " ".join(stemwords)

    return body
```

18.Загрузите коды слов из словаря vocab.txt

In [20]:

```
vocab_data = load_data(DATA_FILE_NAME_6, str, '\t')
vocab = dict(zip(vocab_data[:, 1].tolist(), vocab_data[:, 0].tolist()))
```

19.Реализуйте функцию замены слов в тексте письма после предобработки на их соответствующие коды.

In [21]:

```
def replace_with_codes(body, vocab):
    return set([vocab[word] for word in body.split(' ') if word in vocab])
```

20.Реализуйте функцию преобразования текста письма в вектор признаков (в таком же формате как в файлах spamTrain.mat и spamTest.mat).

In [22]:

```
def transform(text_codes, vocab):
    values_vector = np.zeros(len(vocab), dtype=int)

    for i, code in enumerate(vocab.values()):
        values_vector[i] = int(code in text_codes)

    return values_vector

def build_test_set(emails, vocab, is_processed=False):
    test_set = []

    for email in emails:
        processed_text = email if is_processed else prepare_body(email)
        codes = replace_with_codes(processed_text, vocab)
        values_vector = transform(codes, vocab)
        test_set.append(values_vector)

    return np.array(test_set)
```

21.Проверьте работу классификатора на письмах из файлов emailSample1.txt, emailSample2.txt, spamSample1.txt и spamSample2.txt.

In [23]:

```
spam_classificator = SVC(kernel='rbf', C=C_train, gamma=gamma_train)
spam_classificator.fit(X, Y)


nltk.download("stopwords")
print('\n')

filenames = ['emailSample1', 'emailSample2', 'spamSample1', 'spamSample2']
emails = [open(DATA_DIRECTORY + f'Lab 5/{filename}.txt').read() for filename in filenames]
test_set = build_test_set(emails, vocab)

result = spam_classificator.predict(test_set)

print('Spam classifier prediction:', result)
print('Excpected result:', [0, 0, 1, 1], '\n')
```

```
Spam classifier prediction: [0 0 1 1]
Excpected result: [0, 0, 1, 1]
```

```
[nltk_data] Downloading package stopwords to /home/nox/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

22.Также можете проверить его работу на собственных примерах.

In [24]:

```
filenames = ['emailExample', 'emailSpam', 'emailExample1', 'emailSpam1']
custom_emails = [open(CUSTOM_DATA_DIRECTORY + f'5/{filename}.txt').read() for filename in filenames
]
test_set = build_test_set(custom_emails, vocab)

result = spam_classificator.predict(test_set)

print('Spam classifier prediction:', result)
print('Excpected result:', [0, 1, 0, 1], '\n')
```

```
Spam classifier prediction: [0 1 0 1]
Excpected result: [0, 1, 0, 1]
```

23.Создайте свой набор данных из оригинального корпуса текстов - http://spamassassin.apache.org/old/publiccorpus/.

Загрузим созданный набор данных:

In [25]:

```
spam_path = CUSTOM_DATA_DIRECTORY + f'5/spam'
ham_path = CUSTOM_DATA_DIRECTORY + f'5/easy_ham'

spamfiles = [join(spam_path, fname) for fname in listdir(spam_path)]
hamfiles = [join(ham_path, fname) for fname in listdir(ham_path)]
```

24.Постройте собственный словарь.

Подготовим наши тексты писем к обучению:

In [26]:

```
all_files = hamfiles + spamfiles
emails_processed = [''] * len(spamfiles)
Yreal = [0] * len(hamfiles) + [1] * len(spamfiles)

for i, filename in enumerate(spamfiles):
    try:
        body = prepare_body(open(filename, 'r', encoding='utf-8').read())
    except UnicodeDecodeError:
        continue

    if not body:
        continue

    emails_processed[i] = prepare_body(body)
```

Создадим свой словарь:

In [27]:

```
# build vocab
all_words = [word for email in emails_processed for word in email.split(" ")]
words_counter = collections.Counter(all_words)

words_list = [word for word in words_counter if words_counter[word] > 100 and len(word) > 1]
custom_vocab = {word: i for i, word in enumerate(words_list)}
```

25. Как изменилось качество классификации? Почему?

In [28]:

```
emails = [''] * len(hamfiles)
for i, filepath in enumerate(hamfiles):
    try:
        emails[i] = prepare_body(open(filename, 'r', encoding='utf-8').read())
    except UnicodeDecodeError:
        continue

emails += emails_processed
real_test_set = build_test_set(emails, custom_vocab, is_processed=True)

real_classifier = SVC(kernel='rbf', C=C_train, gamma=gamma_train)
real_classifier.fit(real_test_set, Yreal)

print(f'Real classicator score: {real_classifier.score(real_test_set, Yreal)}')
print(f'Test classicator score: {spam_classificator.score(Xtest, Ytest)}')
```

```
Real classicator score: 0.9992401215805471
Test classicator score: 0.988
```

По результатам видно, что точность классификации для тестовой выборки немного выше. Это связано с тем, что количество слов в тестовом словаре больше. А значит в для одного и того же документа можно составить два разных вектора, в одном из которых среднее количество вхождений слов из словаря будет больше.