Отчёт по лабораторной работе №5 "Метод опорных векторов"

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import warnings
warnings.filterwarnings('ignore')
import re
import html
import nltk
from nltk.stem.snowball import SnowballStemmer
from sklearn.svm import SVC
from nltk.corpus import stopwords
nltk.download("stopwords")

%matplotlib inline
```

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

In [2]:
```
from scipy.io import loadmat

def load_file(filename, keys=None):
    if keys is None:
        keys = ['X', 'y']
    mat = loadmat(filename)
    ret = tuple([mat[k].reshape(mat[k].shape[0]) if k.startswith('y') else mat[k] for k in keys])
    return ret
```

1. Загрузите данные ex5data1.mat из файла.

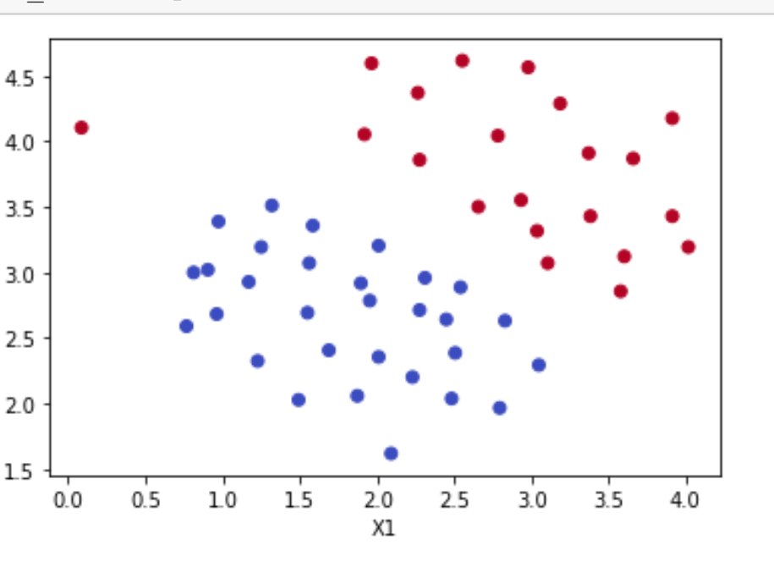
In [3]:
```
X, y = load_file('ex5data1.mat')
print(f'Train shape: {X.shape, y.shape}')
```

```
Train shape: ((51, 2), (51,))
```

2. Постройте график для загруженного набора данных: по осям - переменные X1, X2, а точки, принадлежащие различным классам должны быть обозначены различными маркерами.

In [4]:
```
def plot_data(X, y):
    z_true = X[y == 1]
    z_false = X[y == 0]
    fig, ax = plt.subplots()

    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    plt.show()
```

In [5]:
```
plot_data(X, y)
```



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

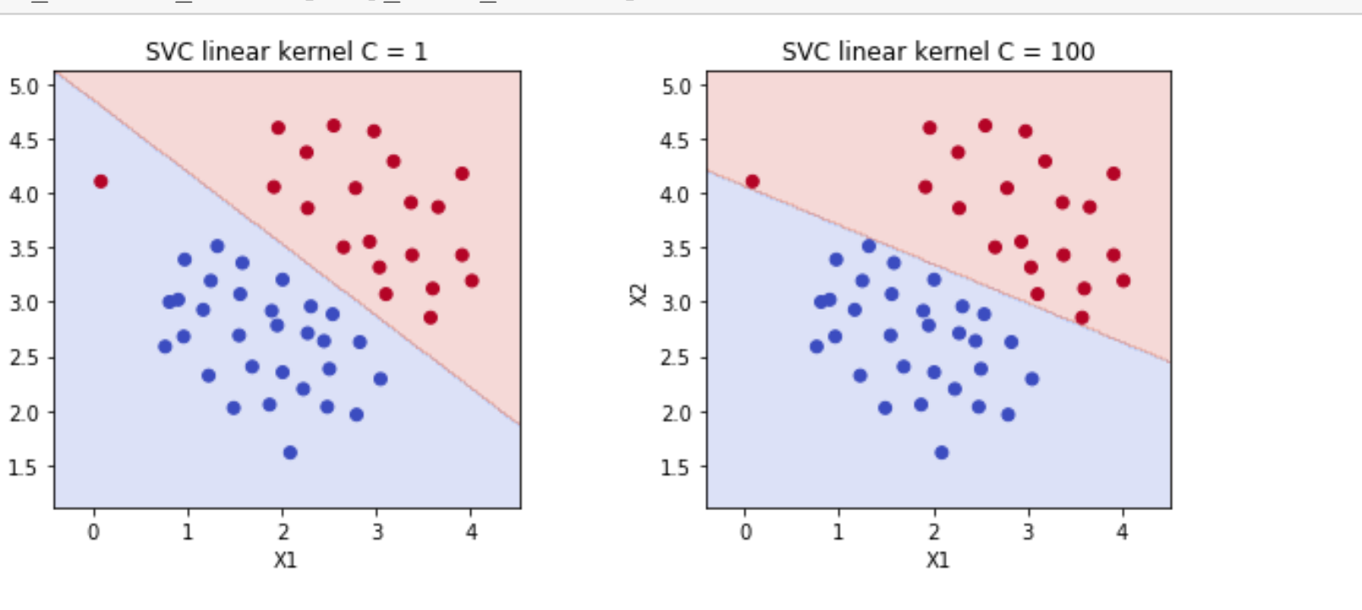
In [6]:
```
clf_c1 = SVC(kernel='linear', C=1.0)
clf_c1.fit(X, y)

clf_c100 = SVC(kernel='linear', C=100.0)
clf_c100.fit(X, y)
```

Out[6]:
```
SVC(C=100.0, 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 [7]:
```
def plot_decision_boundary(map_title_clf, 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[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    plt.figure(figsize=(10, 4))

    for i, title in enumerate(map_title_clf.keys()):
        plt.subplot(1, 2, i + 1)
        plt.subplots_adjust(wspace=0.4, hspace=0.4)
        Z = map_title_clf[title].predict(np.c_[xx.ravel(), yy.ravel()])

        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=cm.coolwarm, alpha=0.2, **kwargs['contour_params'])

        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cm.coolwarm, **kwargs['scatter_params'])
        plt.xlabel('X1')
        plt.ylabel('X2')
        plt.xlim(X[:, 0].min() - bias, X[:, 0].max() + bias)
        plt.ylim(X[:, 1].min() - bias, X[:, 1].max() + bias)
        plt.title(title)

    plt.show()
```

In [8]:
```
map_title_clf = {
    'SVC linear kernel C = 1': clf_c1,
    'SVC linear kernel C = 100': clf_c100,
}
plot_decision_boundary(map_title_clf, X, y)
```



Из графиков можно заметить, то при C = 100 разделяющая кривая смещается в сторону одного, отклоняющегося от всех примера. Это происходит потому, что чем больше коефицент C, тем модель более чувствительна (high variance), то есть больше склонна к переобучению. А вот C = 1 - достаточно удачный параметр регуляризации, так как модель не придаёт большой важности одному отклоняющемуся примеру.

5. Реализуйте функцию вычисления Гауссового ядра для алгоритма SVM.

In [9]:
```
def gaussian_kernel(sigma):
    def wrapped(x, l):
        degree = ((x - l)**2).sum(axis=1)
        return np.e ** (-degree) / (2 * sigma**2)

    return wrapped
```

6. Загрузите данные ex5data2.mat из файла.

In [10]:
```
X, y = load_file('ex5data2.mat')
print(f'Train shape: {X.shape, y.shape}')
```

```
Train shape: ((863, 2), (863,))
```

7. Обработайте данные с помощью функции Гауссового ядра.

In [11]:
```
kernel_func = gaussian_kernel(1)

def process_input_with_gaus(X, *args):
    return np.array([kernel_func(X, l) for l in X])
```

In [12]:
```
X_gaussian = process_input_with_gaus(X)
```

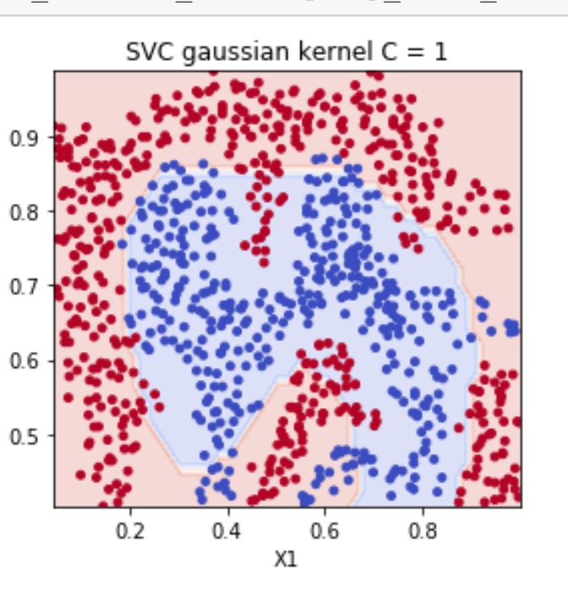
8. Обучите классификатор SVM.

In [13]:
```
clf_gaussian = SVC(kernel='rbf', C=1, gamma='scale')
clf_gaussian.fit(X, y)
```

Out[13]:
```
SVC(C=1, 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)
```

9. Визуализируйте данные вместе с разделяющей кривой.

In [14]:
```
map_title_clf = {'SVC gaussian kernel C = 1': clf_gaussian}
plot_decision_boundary(map_title_clf, X, y, bias=0.0, scatter_params={'s': 15})
```



10. Загрузите данные ex5data3.mat из файла.

In [15]:
```
X, y, Xval, yval = load_file('ex5data3.mat', keys=['X', 'y', 'Xval', 'yval'])
print(f'Train shape: {X.shape, y.shape}')
print(f'Val shape: {X.shape, y.shape}')
```

```
Train shape: ((211, 2), (211,))
Val shape: ((211, 2), (211,))
```

11. Вычислите параметры классификатора SVM на обучающей выборке, а также подберите параметры C и σ2 на валидационной выборке.

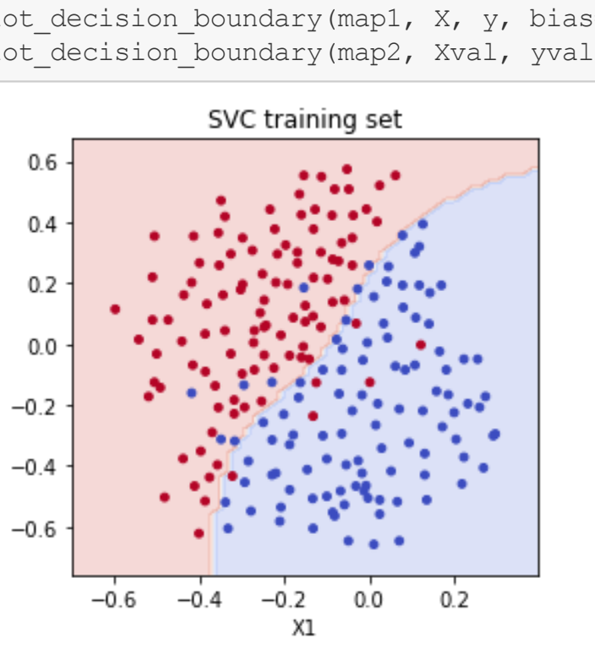
In [16]:
```
def search_optimal(X, y, Xval, yval, C_list, gamma_list):
    best_score = -np.inf
    best_params = None
    for C in C_list:
        for gamma in gamma_list:
            s = SVC(kernel='rbf', C=C, gamma=gamma)
            s.fit(X, y)
            score = s.score(Xval, yval)
            if score > best_score:
                best_score = score
                best_params = (C, gamma)
    return best_params
```

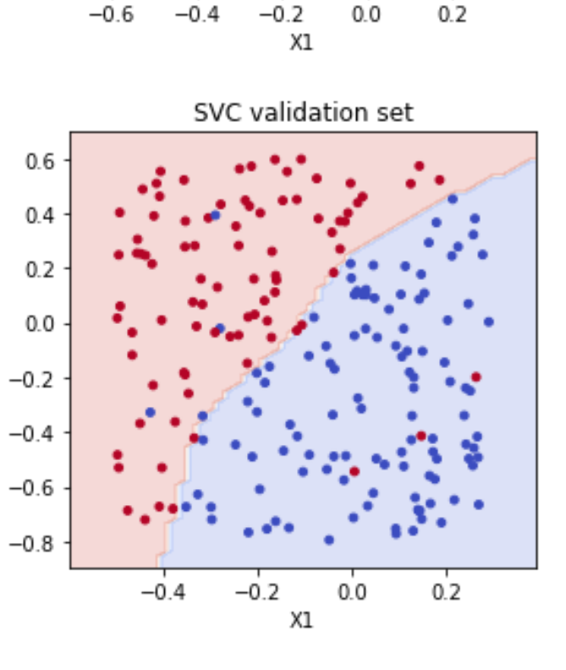
In [17]:
```
best_params = search_optimal(X, y, Xval, yval,
                             C_list=np.logspace(-1, 3, 100), gamma_list=np.linspace(0.0001, 10, 100
))
C_train, gamma_train = best_params
sigma_train = 1 / (2 * gamma_train)
print(f'Best params for validation set: C = {C_train}, sigma squared = {sigma_train}')
```

```
Best params for validation set: C = 0.36783797718286343, sigma squared = 0.05380430688802737
```

12. Визуализируйте данные вместе с разделяющей кривой.

In [18]:
```
C_, gamma_ = best_params
svc_train = SVC(kernel='rbf', C=C_, gamma=gamma_)
svc_train.fit(X, y)
svc_val = SVC(kernel='rbf', C=C_, gamma=gamma_)
svc_val.fit(Xval, yval)
map1 = {
    f'SVC training set': svc_train,
}
map2 = {
    f'SVC validation set': svc_val
}
plot_decision_boundary(map1, X, y, bias=0.1, scatter_params={'s': 15})
plot_decision_boundary(map2, Xval, yval, bias=0.1, scatter_params={'s': 15})
```





13. Загрузите данные spamTrain.mat из файла.

In [20]:
```
X, y = load_file('spamTrain.mat')
print(f'Train shape: {X.shape, y.shape}')
```

```
Train shape: ((4000, 1899), (4000,))
```

14. Обучите классификатор SVM.

In [21]:
```
svm_spam_train = SVC(kernel='rbf')
svm_spam_train.fit(X, y)
```

Out[21]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    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 [23]:
```
Xtest, ytest = load_file('spamTest.mat', keys=['Xtest', 'ytest'])
print(f'Test shape: {Xtest.shape, ytest.shape}')
```

```
Test shape: ((1000, 1899), (1000,))
```

16. Подберите параметры C и σ2.

In [24]:
```
best_params = search_optimal(
    X, y, Xtest, ytest,
    C_list=np.logspace(2, 3, 10), gamma_list=np.linspace(0.0001, 0.0003, 10)
)
print(f'Best params: C = {best_params[0]}, sigma squared = {1 / (2 * best_params[1])}')
```

```
Best params: C = 100.0, sigma squared = 2368.421052631579
```

17. Реализуйте функцию предобработки текста письма.

In [25]:
```
def preprocess(body):
    body = body.lower()

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

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

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

    regx = re.compile(r"\b[\d.]+\b")
    body = regx.sub(repl=" number ", string=body)

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

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

    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 [26]:
```
def load_vocabulary():
    vocab = {}

    with open('vocab.txt', 'r') as f:
        for line in f.readlines():
            l = line.replace('\n', '').split('\t')
            vocab[l[1]] = int(l[0])

    return vocab
```

In [27]:
```
train_vocab = load_vocabulary()
```

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

In [28]:
```
def replace_with_codes(body, vocab):
    return [vocab[word] for word in body.split(' ') if vocab.get(word, None) is not None]
```

20. Реализуйте функцию преобразования текста письма в вектор признаков.

In [29]:
```
from collections import Counter

def transform(codes, vocab):
    codes = set(codes)
    vec = np.zeros(len(vocab), dtype=int)

    for word_code in vocab.values():
        vec[word_code - 1] = int(word_code in codes)

    return vec
```

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

In [30]:
```
def build_test_set(emails, vocab, is_processed=False):
    test_set = []

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

    return np.array(test_set)
```

In [31]:
```
filenames = ['emailSample1.txt', 'emailSample2.txt', 'spamSample1.txt', 'spamSample2.txt']
emails = [open(file).read() for file in filenames]
test_set = build_test_set(emails, train_vocab)
svm_spam = SVC(kernel='rbf', C=best_params[0], gamma=best_params[1])
svm_spam.fit(X, y)
svm_spam.predict(test_set)
```

Out[31]:
```
array([0, 0, 1, 1], dtype=uint8)
```

Из результатов видим, что два первых имейла были классифицированы как не спам, а два последних - как спам, что является ожидаемым результатом.

22. Также можете проверить его работу на собственных примерах.

In [32]:
```
filenames = ['emailMyExample.txt', 'emailMySpam.txt']
emails_examples = [open(file).read() for file in filenames]
example_test_set = build_test_set(emails_examples, train_vocab)
svm_spam_example = SVC(kernel='rbf', C=best_params[0], gamma=best_params[1])
svm_spam_example.fit(X, y)
svm_spam_example.predict(example_test_set)
```

Out[32]:
```
array([0, 1], dtype=uint8)
```

Первый имейл классифицирован как не спам, второе - как спам, что является ожидаемым результатом.

23. Создайте свой набор данных из оригинального корпуса текстов.

In [33]:
```
def get_body(fpath):
    with open(fpath, "r") as f:
        try:
            lines = f.readlines()
            idx = lines.index("\n")
            return "".join(lines[idx:])
        except:
            pass
```

In [34]:
```
import os
from os import listdir
from os.path import isfile, join

spampath = join(os.getcwd(), "spam")
hampath = join(os.getcwd(), "easy_ham")

spamfiles = [join(spampath, fname) for fname in listdir(spampath)]
hamfiles = [join(hampath, fname) for fname in listdir(hampath)]
```

In [35]:
```
all_files = hamfiles + spamfiles
emails_raw =  [''] * len(all_files)
emails_processed =  [''] * len(all_files)
yreal = [0] * len(hamfiles) + [1] * len(spamfiles)  # Ground truth vector

for i, filename in enumerate(all_files):
    body = get_body(filename)
    if body is None:
        continue
    emails_raw[i] = body
    emails_processed[i] = preprocess(body)
```

In [36]:
```
print('==========RAW EMAIL=========')
print(emails_raw[0])
print('=======PROCESSED EMAIL======')
print(emails_processed[0])
```

```
==========RAW EMAIL=========

On Mon, 30 Sep 2002, Tom wrote:

> If the set passes around enough then more people have these works. the
> more folks  that have them now, while they are still legal to have, the
> likely they  will be left behind in the possible/probabale copyright
> chillout..and if  that doesnt happen then more folks than not will still

We will be getting BlackNet-like guerilla P2P pretty soon. Packaging it
into wormcode with an initial userbase of a few 100 k to Mnodes gives you
pretty bulletproof plausible deniability.

> have it for uses all manner of shades.



=======PROCESSED EMAIL======
mon number sep number tom wrote set pass around enough peopl work folk still legal like left beh
ind possibl probabal copyright chillout number doesnt happen folk still get blacknet like gueril
la p2p pretti soon packag wormcod initi userbas number k mnode give pretti bulletproof plausibl
deniabl use manner shade
```

24. Постройте собственный словарь.

In [37]:
```
import collections

all_words = [word for email in emails_processed for word in email.split(" ")]
words_counter = collections.Counter(all_words)
vocab_list = [key for key in words_counter if words_counter[key] > 100 and len(key) > 1]
test_vocab = {word: i for i, word in enumerate(vocab_list)}
print(f'Examples: {vocab_list[:9]}')
```

```
Examples: ['mon', 'number', 'sep', 'wrote', 'set', 'around', 'enough', 'peopl', 'work']
```

In [38]:
```
print(f'Number of words in vocabulary: {len(test_vocab)}')
```

```
Number of words in vocabulary: 651
```

25. Как изменилось качество классификации? Почему?

In [39]:
```
Xreal = build_test_set(emails_processed, test_vocab, is_processed=True)
svm_real = SVC(kernel='rbf', C=best_params[0], gamma=best_params[1])

svm_real.fit(Xreal, yreal)
print(f'Score of real classicator: {svm_real.score(Xreal, yreal)}')
print(f'Score of test classicator: {svm_spam.score(Xtest, ytest)}')
```

```
Score of real classicator: 0.9653564290473018
Score of test classicator: 0.991
```

Можем видеть 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.