Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ЛАБОРАТОРНАЯ РАБОТА №5

«Метод опорных векторов»

Студент А. Ю. Омельчук

Преподаватель М. В. Стержанов

Минск 2019

ХОД РАБОТЫ

Задание.

Набор данных ex5data1.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит три переменные X1 и X2 (независимые переменные) и y (метка класса). Данные являются линейно разделимыми.

Набор данных ex5data2.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит три переменные X1 и X2 (независимые переменные) и y (метка класса). Данные являются нелинейно разделимыми.

Набор данных ex5data3.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит три переменные X1 и X2 (независимые переменные) и y (метка класса). Данные разделены на две выборки: обучающая выборка (X, y), по которой определяются параметры модели; валидационная выборка (Xval, yval), на которой настраивается коэффициент регуляризации и параметры Гауссового ядра.

Набор данных spamTrain.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две переменные X - вектор, кодирующий отсутствие (0) или присутствие (1) слова из словаря vocab.txt в письме, и y - метка класса: 0 - не спам, 1 - спам. Набор используется для обучения классификатора.

Набор данных spamTest.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две переменные Xtest - вектор, кодирующий отсутствие (0) или присутствие (1) слова из словаря vocab.txt в письме, и ytest - метка класса: 0 - не спам, 1 - спам. Набор используется для проверки качества классификатора.

1. Загрузите данные ex5data1.mat из файла.
2. Постройте график для загруженного набора данных: по осям - переменные X1, X2, а точки, принадлежащие различным классам должны быть обозначены различными маркерами.
3. Обучите классификатор с помощью библиотечной реализации SVM с линейным ядром на данном наборе.
4. Постройте разделяющую прямую для классификаторов с различными параметрами C = 1, C = 100 (совместно с графиком из пункта 2). Объясните различия в полученных прямых?
5. Реализуйте функцию вычисления Гауссового ядра для алгоритма SVM.

6. Загрузите данные ex5data2.mat из файла.
7. Обработайте данные с помощью функции Гауссового ядра.
8. Обучите классификатор SVM.
9. Визуализируйте данные вместе с разделяющей кривой (аналогично пункту 4).
10. Загрузите данные ex5data3.mat из файла.
11. Вычислите параметры классификатора SVM на обучающей выборке, а также подберите параметры C и σ2 на валидационной выборке.
12. Визуализируйте данные вместе с разделяющей кривой (аналогично пункту 4).
13. Загрузите данные spamTrain.mat из файла.
14. Обучите классификатор SVM.
15. Загрузите данные spamTest.mat из файла.
16. Подберите параметры C и σ2.
17. Реализуйте функцию предобработки текста письма, включающую 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.
18. Загрузите коды слов из словаря vocab.txt.
19. Реализуйте функцию замены слов в тексте письма после предобработки на их соответствующие коды.
20. Реализуйте функцию преобразования текста письма в вектор признаков (в таком же формате как в файлах spamTrain.mat и spamTest.mat).
21. Проверьте работу классификатора на письмах из файлов emailSample1.txt, emailSample2.txt, spamSample1.txt и spamSample2.txt.
22. Также можете проверить его работу на собственных примерах.

23. Создайте свой набор данных из оригинального корпуса текстов - http://spamassassin.apache.org/old/publiccorpus/.
24. Постройте собственный словарь.
25. Как изменилось качество классификации? Почему?

Результат выполнения:

1. Код выгрузки данных из файла представлен ниже:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex5data1.mat')
dataset = sio.loadmat(file_path)
X = dataset["X"]
y = dataset["y"]
```

2. График представлен ниже:

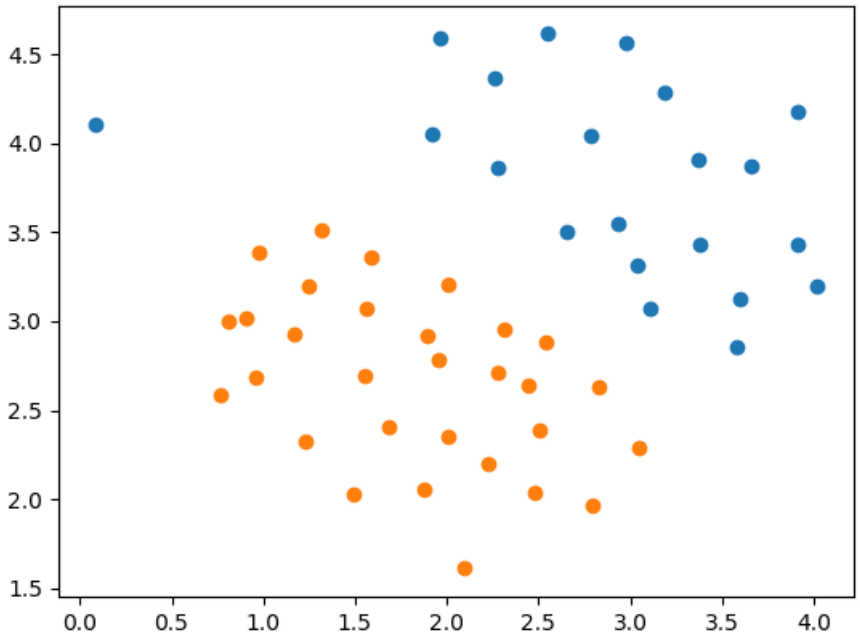

Рисунок 1 – исходные данные

3. Код реализации:

```
classifier = SVC(kernel="linear")
classifier.fit(X, y.flatten())  # default C=1
```

Здесь была использована библиотечная реализация (sklearn) SVC, т. к. в исходном коде на матлабе в курсе от Andrew Ng он сам говорил, что его код считает только аппроксимирующе, и что он всем рекомендует использовать

готовую реализацию, поскольку она считает быстрее и точнее. И, соответственно, переносить все исходные файлы матлаба в питон могло бы просто забрать много времени, поэтому здесь и дальше использована библиотечная реализация.

4. Графики приведены ниже:

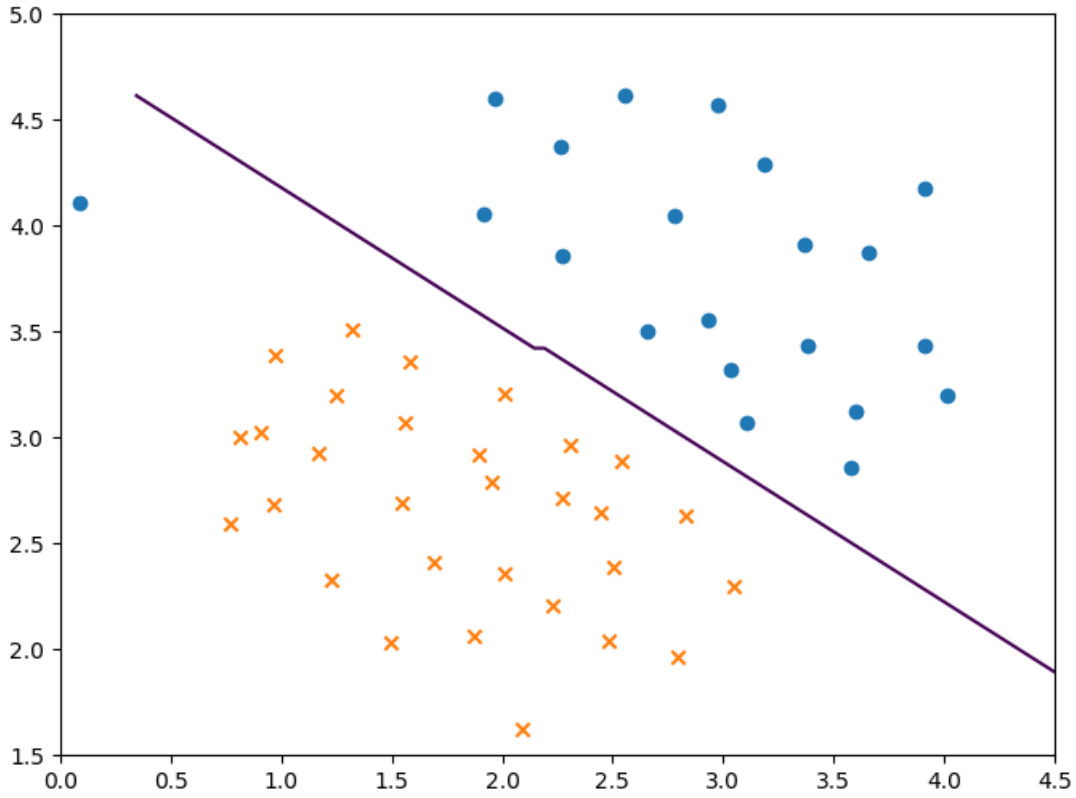

Рисунок 2 – исходные данные и график разделяющей прямой при C=1

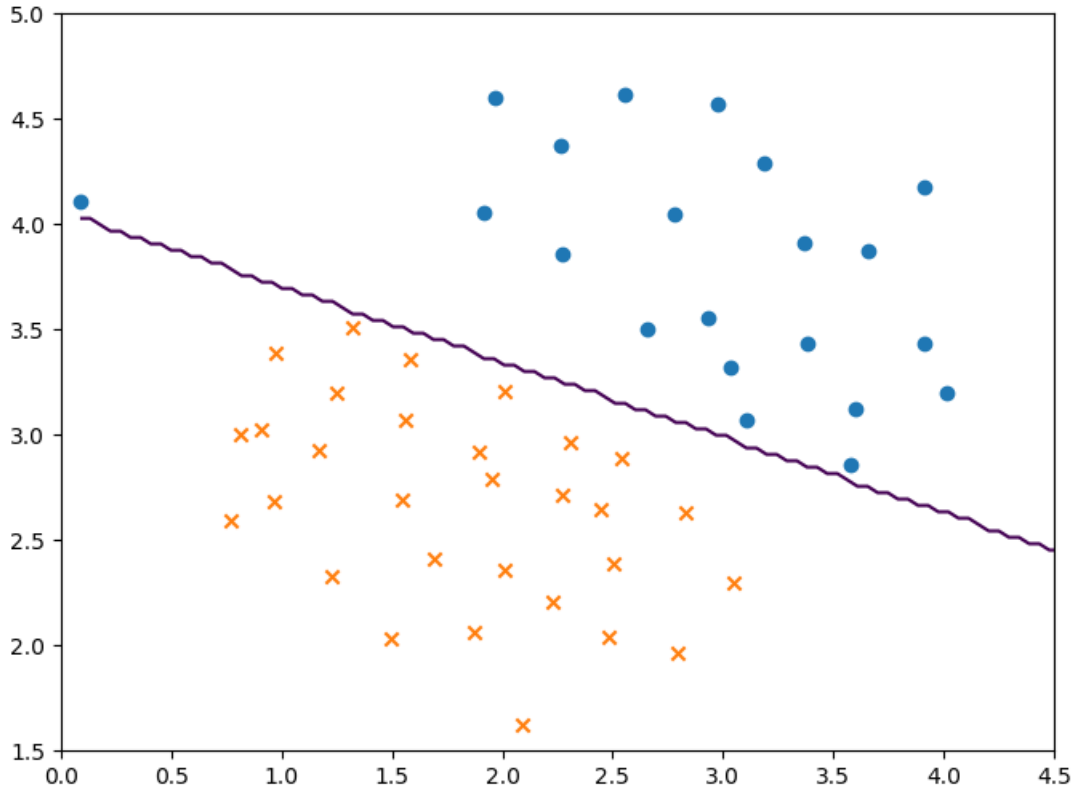

Рисунок 3 – исходные данные и график разделяющей прямой при C=100

Данные графики отличаются, потому что используется различный коэффициент C (1 и 100 сответственно). При коэффициенте равным 1 мы можем видеть, что один экземпляр был засчитан к другому классу и в линии, практически, отсутствуют изгибы, в то время как при коэффициенте 100 все данные были классифицированы правильно.

Функция потерь в SVM преследует две оптимизационные цели: увеличение длинны margin и уменьшение theta. Параметр C контролирует приоритет этих целей.

В первом случае была допущена ошибка в классификации в пользу простоты theta (вторая оптимизационная цель).

Во втором случае с параметром C = 100 цена misclasification возрасла в 100 раз. Соответственно была выбранна такая decision boundary, которая классифицировала все элементы верно, при этом margin уменьшилась. Конечно, в таком случае, шансы того, что наша модель сможет генерализовать и показать столь же хорошие результаты на новых данных, катастрофически мала. Чем выше число C тем более запутанная гиперплоскость будет в модели, но и выше число верно-классифицированных объектов обучающей выборки.

5. Код реализации:

```
def gauss_kernel_carried(sigma):
    def gauss_kernel(x1, x2):
        sigma_squared = np.power(sigma, 2)
        matrix = np.power(x1-x2, 2)

        return np.exp(-np.sum(matrix)/(2*sigma_squared))

    return gauss_kernel

x1 = np.array([1, 2, 1])
x2 = np.array([0, 4, -1])
sigma = 2

sim = gauss_kernel_carried(sigma)(x1, x2)
print('Gaussian Kernel between x1 = [1, 2, 1], x2 = [0, 4, -1], sigma = %0.2f:'
          '\n\t%f\n(for sigma = 2, this value should be about 0.324652)\n' % (sigma, sim))
```

6. Код загрузки данных:

```
    file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex5data2.mat')
    dataset = sio.loadmat(file_path)
    X = dataset["X"]
    y = dataset["y"]
```

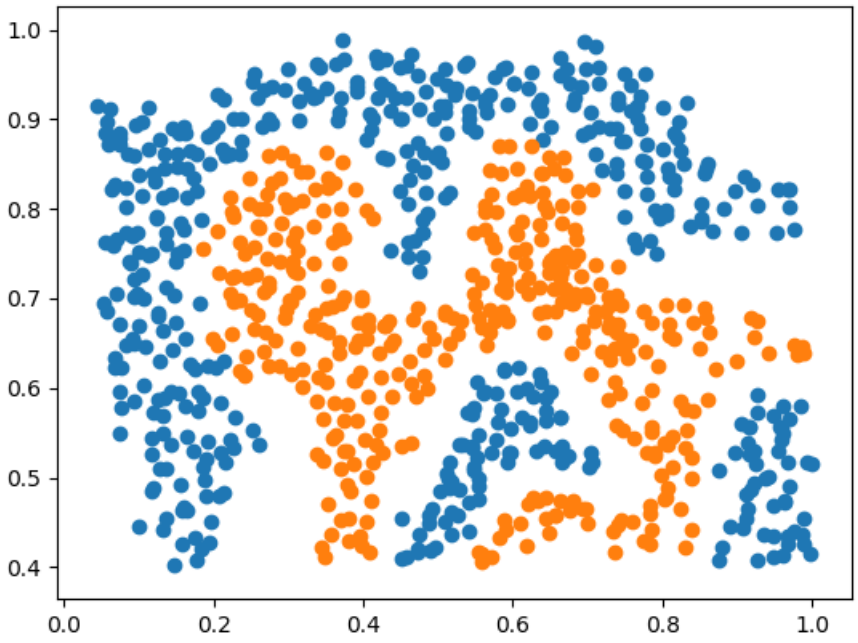

Рисунок 4 – визуализация исходных данных

7. Код реализации:

```
    sigma = 0.1
    kernel = gauss_kernel_carried(sigma)
```

8. Код реализации:

```
gamma = np.power(sigma, -2.)
classifier3 = SVC(C=1, kernel='rbf', gamma=gamma)
classifier3.fit(X, y.flatten())
```

9. График приведён ниже:

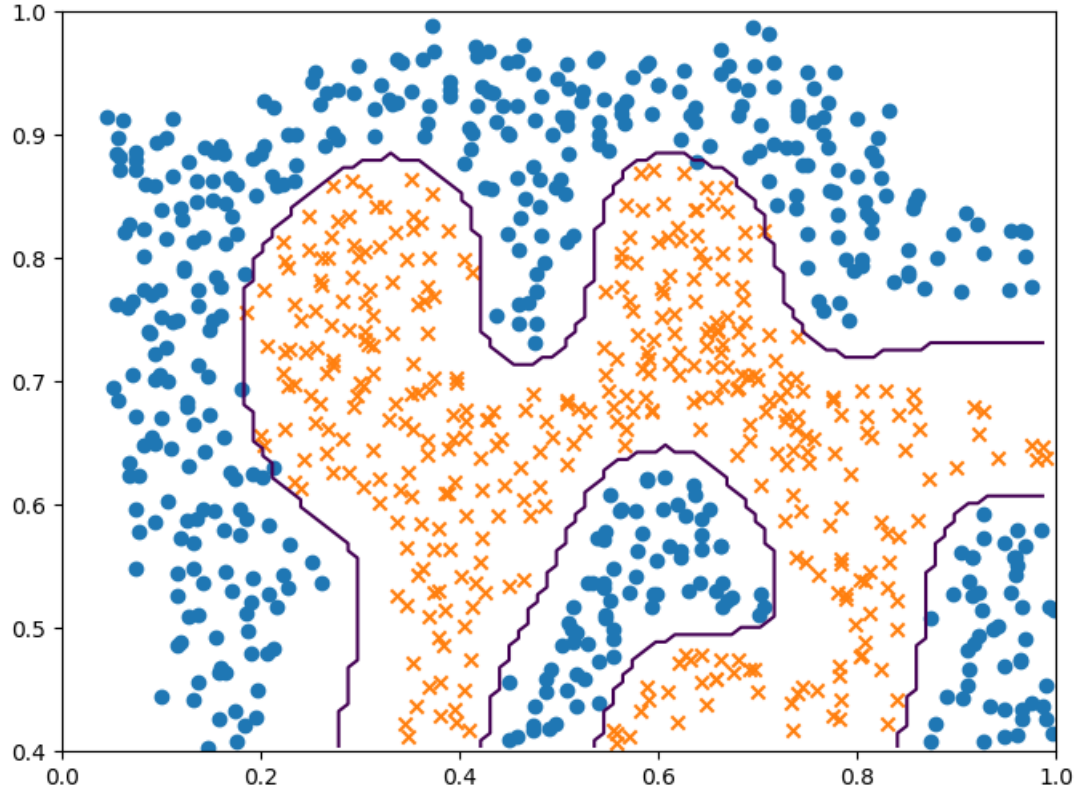

Рисунок 5 – визуализация разделяющей кривой

10. Код выгрузки данных из файла представлен ниже:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex5data3.mat')
dataset = sio.loadmat(file_path)
X = dataset["X"]
y = dataset["y"]
Xval = dataset["Xval"]
yval = dataset["yval"]
```

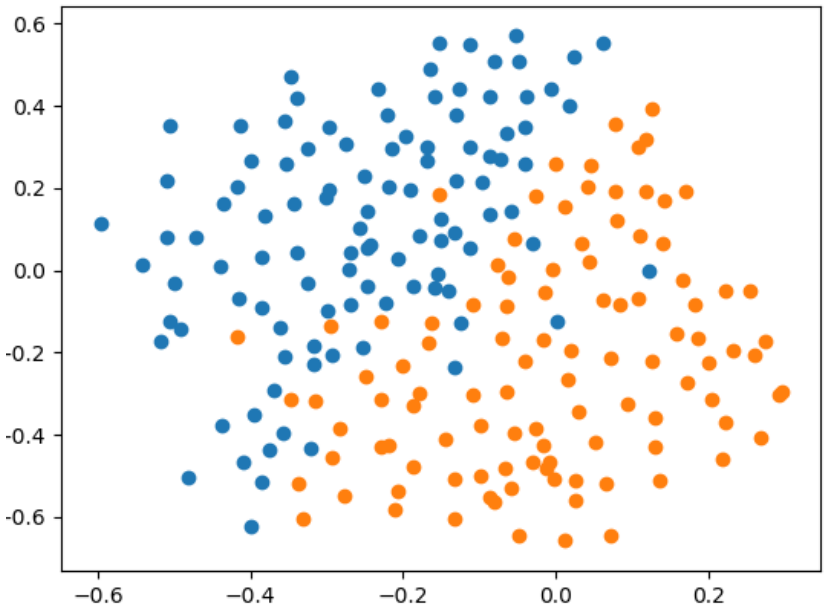

Рисунок 6 – визауализация исходных данных

11. Код реализации:

```
def dataset3Params(X, y, Xval, yval, values):
    # You need to return the following variables correctly.
    C = values[0]
    sigma = values[0]
    result_score = 0
    # ====================== YOUR CODE HERE ======================
    for i in values:
        for j in values:
            gamma = 1 / j
            classifier = SVC(C=i, gamma=gamma, kernel='rbf')
            classifier.fit(X, y)
            prediction = classifier.predict(Xval)
            score = classifier.score(Xval, yval)
            print("i: ", i, "j: ", j, "score: ", score)
            if score > result_score:
                result_score = score
                C = i
                sigma = gamma

    # =============================================================
    return C, sigma

vals = [0.01, 0.03, 0.1, 0.3, 0.5, 1, 3, 10, 30, 50, 100]
C, gamma = dataset3Params(X, y.flatten(), Xval, yval.flatten(), vals)
print("C: ", C, ", gamma: ", gamma)
classifier4 = SVC(C=C, gamma=gamma, kernel='rbf')
classifier4.fit(X, y.flatten())
```

Результат выполнения:

```
('C: ', 0.3, ', gamma: ', 100.0)
```

12. График приведён ниже:

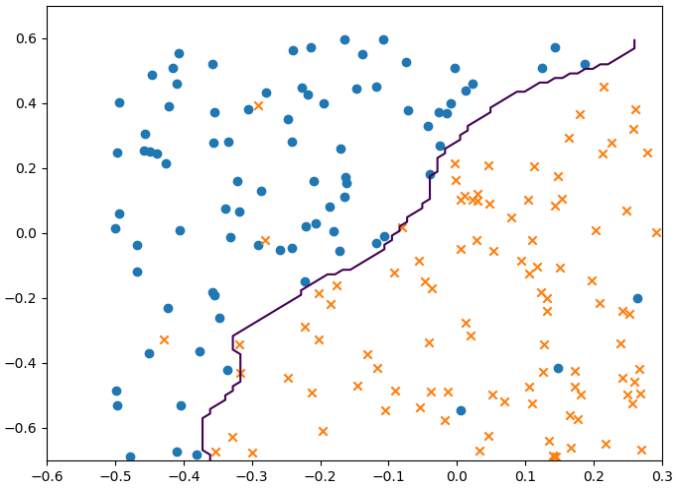

Рисунок 7 – график исходных данных вместе с разделяющей кривой

13. Код выгрузки данных из файла представлен ниже:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'spamTrain.mat')
dataset = sio.loadmat(file_path)
X = dataset["X"]
y = dataset["y"]
```

14. Код реализации:

```
C = 0.1
classifier5 = SVC(C=C, kernel='linear')
classifier5.fit(X, y.flatten())
print('Training Accuracy: ', (classifier5.score(X, y.flatten())) * 100)
```

Результат выполнения:

```
'Training Accuracy: ', 99.825
```

15. Код выгрузки данных из файла представлен ниже:

```
file_path = os.path.join(os.path.dirname(__file__), 'data', 'spamTest.mat')
dataset = sio.loadmat(file_path)
Xtest = dataset["Xtest"]
ytest = dataset["ytest"]
```

16. Для обучения использовался Гауссовский классификатор, для подбора параметров использовался тот же метод, что и в пункте 11. После всех вычисления был получен следующий результат:

```
    C = 30, gamma=0.001 ~ 99.1
```

17. Код реализации:

```
    def process_email(email_contents):
        """
        Preprocesses the body of an email and returns a list of indices of the words
contained in the email.
        """
        # a - Lower case
        email_contents = email_contents.lower()

        # b - remove html/xml tags
        email_contents = re.sub("<[^>]*>", " ", email_contents).split(" ")
        email_contents = filter(len, email_contents)
        email_contents = ' '.join(email_contents)

        # c - Handle URLS
        email_contents = re.sub("[http|https]://[^\s]*", "httpaddr", email_contents)

        # d - Handle Email Addresses
        email_contents = re.sub("[^\s]+@[^\s]+", "emailaddr", email_contents)

        # e - Handle numbers
        email_contents = re.sub("[0-9]+", "number", email_contents)

        # f - Handle $ sign
        email_contents = re.sub("[$]+", "dollar", email_contents)

        # Strip all special characters
        special_chars = [
            "<", "[", "^", ">", "+", "?", "!", "'", ".", ",", ":",
            "*", "%", "#", "_", "="
        ]
        for char in special_chars:
            email_contents = email_contents.replace(str(char), "")
        email_contents = email_contents.replace("\n", " ")

        # Stem the word
        ps = PorterStemmer()
        email_contents = [ps.stem(token) for token in email_contents.split(" ")]
        email_contents = " ".join(email_contents)

        return email_contents
```

18. Код выгрузки данных:

```
vocabList = open(os.path.join(os.path.dirname(__file__), 'data', 'vocab2.txt'), "r").read()
    vocabList = vocabList.split("\n")
    vocabList_d = {}
    for ea in vocabList:
        [value, key] = ea.split("\t")
        vocabList_d[key] = value
```

19. Код функции:

```
def find_word_indices(processed_email, vocabList_d):
    # Process the email and return word_indices

    word_indices = []

    for char in processed_email.split():
        if len(char) > 1 and char in vocabList_d:
            word_indices.append(int(vocabList_d[char]))

    return word_indices
```

20. Код реализации:

```
def email_features(word_indices, vocabList_d):
    """
    Takes in a word_indices vector and  produces a feature vector from the word indices.
    """
    n = len(vocabList_d)

    features = np.zeros((n, 1))

    for i in word_indices:
        features[i] = 1

    return features
```

21. Код реализации:

```
        email_sample1       =       open(os.path.join(os.path.dirname(__file__),       'data',
'emailSample1.txt'), "r").read()
        email_sample2       =       open(os.path.join(os.path.dirname(__file__),       'data',
'emailSample2.txt'), "r").read()
        spam_sample1 = open(os.path.join(os.path.dirname(__file__), 'data', 'spamSample1.txt'),
"r").read()
        spam_sample2 = open(os.path.join(os.path.dirname(__file__), 'data', 'spamSample2.txt'),
"r").read()

        email_sample1 = transform_email_to_features(email_sample1, vocabList_d)
        email_sample2 = transform_email_to_features(email_sample2, vocabList_d)
        spam_sample1 = transform_email_to_features(spam_sample1, vocabList_d)
        spam_sample2 = transform_email_to_features(spam_sample2, vocabList_d)

        print('Spam  -> 1\nEmail -> 0')
        print('\n Gaussian Kernel: ')
        print('False', classifier6.predict(email_sample1.T))
        print('False', classifier6.predict(email_sample2.T))
        print('Spam', classifier6.predict(spam_sample1.T))
        print('Spam', classifier6.predict(spam_sample2.T))
        print('\n')
```

Результат выполнения:

```
    Spam  -> 1
```

```
Email -> 0
Gaussian Kernel:
('False', array([0], dtype=uint8))
('False', array([0], dtype=uint8))
('Spam', array([0], dtype=uint8))
('Spam', array([1], dtype=uint8))
```

Как видно из результатов, семплы емейлов были классифицированы правильно, однако одно из спам-сообщений было помечено как емейл-сообщение. В реализации матлаба, однако, всё было классифицировано правильно. Я думаю, что здесь сыграло то, что использовалась библиотечная реализации, вместо той, что использует Andrew Ng в своём курсе в программе матлаб.

22. Код реализации:

```
        spam_sample = "Hi, Kirill Zyusko, Items on your wishlist are now discounted! Some items you've added to your personal wishlist are currently discounted on http://www.gog.com/ - you can find all the details below or directly on your https://www.gog.com/account/wishlist."
        spam_sample = transform_email_to_features(spam_sample, vocabList_d)
        print('\n Gaussian Kernel: ')
        print('Spam', classifier6.predict(spam_sample.T))

        email_sample = "Hi Kirill Zyusko, Your job search status is open, but not actively looking, but we need more information before accelerating your matches. Once you tell us just a little bit more about yourself, we'll get you in front of companies and send you any new jobs that match your interests."
        email_sample = transform_email_to_features(email_sample, vocabList_d)
        print('Email (should be false - at least gmail doesn\'t classify this message as a spam)', classifier6.predict(email_sample.T))

        spam_sample3 = "Hi, Hope you are doing great. This is Josh from Avco Consulting Inc. Inc is a global IT company based in Worcester, MA. Our leadership in the industry has been established by our excellence in helping clients use Information Technology to achieve their business objectives. Our core competencies are Information Technology (IT)services and Project Management. I have available consultants with Excellent communication, analytical, and team work skills. I would appreciate if you - or someone you can recommend share the suitable requirements accordingly. Below is the list of the available consultant with different skills set and their preferred locations."
        spam_sample3 = transform_email_to_features(spam_sample3, vocabList_d)
        print('Spam', classifier6.predict(spam_sample3.T))
```

Результат выполнения:

```
    ('Spam', array([1], dtype=uint8))
    ("Email (should be false - at least gmail doesn't classify this message as a spam)", array([0], dtype=uint8))
    ('Spam', array([1], dtype=uint8))
```

23. Данные были скачаны по ссылке, что была дана в задании. Я решил построить свой классификатор на основе двух файлов:

20030228_easy_ham.tar.bz2 и 20030228_spam.tar.bz2. Выбор пал именно на эти два файла, поскольку, судя по дате они были загружены в одно и то же время, и значит, что содержат, примерно целостную информацию. Файл со спамом содержит порядка 500 сообщений, файлы не спама содержат около 2500 сообщений. Перед тем как приступить к подготовке данных был произведён анализ их заголовков. В ходе этого анализа было выяснено, что почти все сообщения представлены в text/html формате, что означало, что данные сообщения можно было обрабатывать дальше. После этого была произведена фильтрация, а именно, из исходных данных были вырезаны заголовки, информация о отправителе и пр., чтобы как можно ближе приблизиться к исходной задаче в лабораторной, т. е. к тем данным, на которых классификатор обучался ранее. После того, как контент сообщений был получен их нужно было подготовить к дальнейшей обработке. Были проведены все те же этапы – удаление html-тегов, стэмминг, перевод в нижний регистр, удаление лишних символов и замена специальных. После этого в спам сообщениях были найдено 1899 наиболее встречающихся слов. В дальнейшем эти слова были отсортированы в алфавитном порядке и записаны в специальный файл.

Также исходные данные были разделены по правилу 80/20 на тренировочную и тестовую выборки.

24. Семпл словарей в сравнение с оригинальным словарём от Andrew Ng:

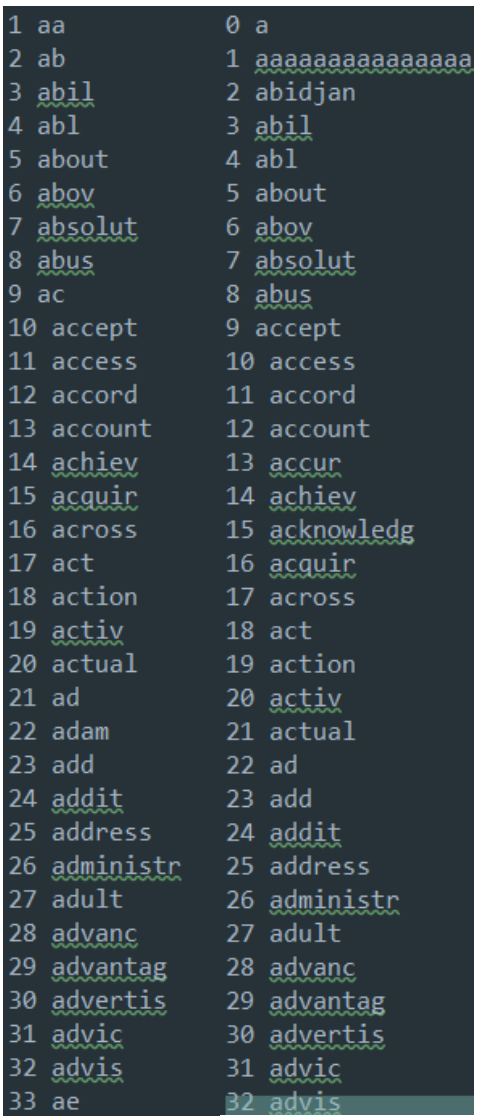

Рисунок 8 – Наглядное сравнение двух словарей (слева – Andrew Ng, справа – мой)

25. Результаты обучения:

```
('Training Accuracy: ', 99.54166666666666)
('Test Accuracy (linear):', 98.82747068676717, '%')
('Training Accuracy (gaussian):', 99.33333333333333, '%')
('Test Accuracy (gaussian):', 98.99497487437185, '%')
```

Модель обучилась очень хорошо. Рассмотрим её поведение на данных, данные в этой лабораторной работе:

```
Spam  -> 1
Email -> 0
Linear Kernel:
('False', array([0]))
('False', array([0]))
```

```
('Spam', array([0]))
('Spam', array([1]))

 Gaussian Kernel:
('False', array([0]))
('False', array([0]))
('Spam', array([1]))
('Spam', array([1]))
```

Видно, что модель с гауссовским ядром отнесла все примеры безошибочно, в то время как модель с линейным ядром допустила одну ошибку.

Однако на моих собственных данных получились следующие результаты:

```
Gaussian Kernel:
('Spam', array([0]))
("Email (should be false - at least gmail doesn't classify this message as a spam)", array([0]))
('Spam', array([0]))
```

Т. е. весь спам пошёл как не спам, но в то же время и настоящее письмо не было распознано как спам, что тоже довольно хороший результат.

Очевиден тот факт, что правильная классификация очень сильно зависит от того, на каких данных обучалась модель.

Программный код:

```
from __future__ import division
from goto import with_goto
import scipy.io as sio
import matplotlib.pyplot as plt
import os
import numpy as np
from sklearn.svm import SVC
import re
from nltk.stem import PorterStemmer


def decision_boundary(classifier, X, y, xlim_min=0, xlim_max=4.5, ylim_min=1.5, ylim_max=5):
        m, n = X.shape[0], X.shape[1]
        pos, neg = (y == 1).reshape(m, 1).flatten(), (y == 0).reshape(m, 1).flatten()
        plt.figure(figsize=(8, 6))
        plt.scatter(X[pos, 0], X[pos, 1])
        plt.scatter(X[neg, 0], X[neg, 1], marker="x")
        # plotting the decision boundary
        X_1, X_2 = np.meshgrid(
            np.linspace(X[:, 0].min(), X[:, 1].max(), num=100),
            np.linspace(X[:, 1].min(), X[:, 1].max(), num=100)
        )
        plt.contour(X_1, X_2, classifier.predict(np.array([X_1.flatten(), X_2.flatten()]).T).reshape(X_1.shape), 1)
```

```
        plt.xlim(xlim_min, xlim_max)
        plt.ylim(ylim_min, ylim_max)

        plt.show()


    def gauss_kernel_carried(sigma):
        def gauss_kernel(x1, x2):
            sigma_squared = np.power(sigma, 2)
            matrix = np.power(x1-x2, 2)

            return np.exp(-np.sum(matrix)/(2*sigma_squared))

        return gauss_kernel


    def plot_data(X, y):
        m, n = X.shape[0], X.shape[1]
        pos, neg = (y == 1).reshape(m, 1).flatten(), (y == 0).reshape(m, 1).flatten()
        plt.scatter(X[pos, 0], X[pos, 1])
        plt.scatter(X[neg, 0], X[neg, 1])
        plt.show()


    def dataset3Params(X, y, Xval, yval, values):
        # You need to return the following variables correctly.
        C = values[0]
        sigma = values[0]
        result_score = 0
        # ====================== YOUR CODE HERE ======================
        for i in values:
            for j in values:
                gamma = 1 / j
                classifier = SVC(C=i, gamma=gamma, kernel='rbf')
                classifier.fit(X, y)
                prediction = classifier.predict(Xval)
                score = classifier.score(Xval, yval)
                print("i: ", i, "j: ", j, "score: ", score)
                if score > result_score:
                    result_score = score
                    C = i
                    sigma = gamma

        # =============================================================
        return C, sigma


    def process_email(email_contents):
        """
        Preprocesses the body of an email and returns a list of indices of the words contained
in the email.
        """
        # a - Lower case
        email_contents = email_contents.lower()

        # b - remove html/xml tags
        email_contents = re.sub("<[^>]*>", " ", email_contents).split(" ")
        email_contents = filter(len, email_contents)
        email_contents = ' '.join(email_contents)
```

```
    # c - Handle URLS
    email_contents = re.sub("[http|https]://[^\s]*", "httpaddr", email_contents)

    # d - Handle Email Addresses
    email_contents = re.sub("[^\s]+@[^\s]+", "emailaddr", email_contents)

    # e - Handle numbers
    email_contents = re.sub("[0-9]+", "number", email_contents)

    # f - Handle $ sign
    email_contents = re.sub("[$]+", "dollar", email_contents)

    # Strip all special characters
    special_chars = [
        "<", "[", "^", ">", "+", "?", "!", "'", ".", ",", ":",
        "*", "%", "#", "_", "="
    ]
    for char in special_chars:
        email_contents = email_contents.replace(str(char), "")
    email_contents = email_contents.replace("\n", " ")

    # Stem the word
    ps = PorterStemmer()
    email_contents = [ps.stem(token) for token in email_contents.split(" ")]
    email_contents = " ".join(email_contents)

    return email_contents


def find_word_indices(processed_email, vocabList_d):
    # Process the email and return word_indices

    word_indices = []

    for char in processed_email.split():
        if len(char) > 1 and char in vocabList_d:
            word_indices.append(int(vocabList_d[char]))

    return word_indices


def transform_email_to_features(email_contents, vocabList_d):
    processed_email = process_email(email_contents)
    word_indices = find_word_indices(processed_email, vocabList_d)
    features = email_features(word_indices, vocabList_d)

    return features


def email_features(word_indices, vocabList_d):
    """
    Takes in a word_indices vector and  produces a feature vector from the word indices.
    """
    n = len(vocabList_d)

    features = np.zeros((n, 1))

    for i in word_indices:
```

```
            features[i] = 1

        return features


    @with_goto
    def main():
        goto .task
        label .task

        # 1
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex5data1.mat')
        dataset = sio.loadmat(file_path)
        X = dataset["X"]
        y = dataset["y"]

        # 2
        # [:, 0] - equals to flatten
        plot_data(X, y)

        # 3
        classifier = SVC(kernel="linear")
        classifier.fit(X, y.flatten())  # default C=1

        # 4
        decision_boundary(classifier, X, y)
        # Test C = 100
        classifier2 = SVC(C=100, kernel="linear")  # gives a decision boundary that overfits
the training examples
        classifier2.fit(X, y.flatten())
        decision_boundary(classifier2, X, y)

        # 5
        x1 = np.array([1, 2, 1])
        x2 = np.array([0, 4, -1])
        sigma = 2

        sim = gauss_kernel_carried(sigma)(x1, x2)
        print('Gaussian Kernel between x1 = [1, 2, 1], x2 = [0, 4, -1], sigma = %0.2f:'
              '\n\t%f\n(for sigma = 2, this value should be about 0.324652)\n' % (sigma, sim))

        # 6
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex5data2.mat')
        dataset = sio.loadmat(file_path)
        X = dataset["X"]
        y = dataset["y"]

        plot_data(X, y)

        # 7
        sigma = 0.1
        kernel = gauss_kernel_carried(sigma)

        # 8
        gamma = np.power(sigma, -2.)
        classifier3 = SVC(C=1, kernel='rbf', gamma=gamma)
        classifier3.fit(X, y.flatten())

        # 9
```

```
        decision_boundary(classifier3, X, y, 0, 1, .4, 1)

        # 10
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex5data3.mat')
        dataset = sio.loadmat(file_path)
        X = dataset["X"]
        y = dataset["y"]
        Xval = dataset["Xval"]
        yval = dataset["yval"]

        plot_data(X, y)
        # plot_data(Xval, yval)

        # 11
        vals = [0.01, 0.03, 0.1, 0.3, 0.5, 1, 3, 10, 30, 50, 100]
        C, gamma = dataset3Params(X, y.flatten(), Xval, yval.flatten(), vals)
        print("C: ", C, ", gamma: ", gamma)
        classifier4 = SVC(C=C, gamma=gamma, kernel='rbf')
        classifier4.fit(X, y.flatten())

        # 12
        decision_boundary(classifier4, Xval, yval, -0.6, 0.3, -0.7, 0.7)

        # 13
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'spamTrain.mat')
        dataset = sio.loadmat(file_path)
        X = dataset["X"]
        y = dataset["y"]

        # 14
        C = 0.1
        classifier5 = SVC(C=C, kernel='linear')
        classifier5.fit(X, y.flatten())
        print('Training Accuracy: ', (classifier5.score(X, y.flatten())) * 100)

        # 15
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'spamTest.mat')
        dataset = sio.loadmat(file_path)
        Xtest = dataset["Xtest"]
        ytest = dataset["ytest"]

        # 16
        print("Test Accuracy (linear):", (classifier5.score(Xtest, ytest.flatten()))*100, "%")
        # C = 1, gamma=0.01 ~98.7
        # C = 30, gamma=0.001 ~ 99.1
        classifier6 = SVC(C=30, kernel='rbf', gamma=0.001)
        classifier6.fit(X, y.flatten())
        print("Training Accuracy (gaussian):", (classifier6.score(X, y.flatten()))*100, "%")
        print("Test  Accuracy  (gaussian):",  (classifier6.score(Xtest,  ytest.flatten()))*100,
"%")


        # 17
        # simple test
        assert  process_email("Hello  WoRlD")  ==  "hello  world",  'Not  implemented  toLowerCase
functional'
        assert                  process_email("<i                  class=\"italic\"><p><b>Hello
world</b><span>Ola</span></p></i>") == "hello world ola", 'Not implemented removing html-tags
functional'
```

```
        assert process_email("http://www.leningrad.spb.ru") == "htthttpaddr", 'Not implemented
replacing http addresses functional'
        assert  process_email("zyusko.kirik@gmail.com")  ==  "emailaddr",  'Not  implemented
replacing email addresses functional'
        assert process_email("amount is 5334$") == "amount is numberdollar", 'Not implemented:
replacing $ functional'
        assert process_email("discounted") == "discount", 'Stemming not implemented'
        assert process_email("*you won* ^_^") == "you won ", 'Removing special characters not
implemented'

        # 18
        vocabList  =  open(os.path.join(os.path.dirname(__file__),  'data',  'vocab2.txt'),
"r").read()
        vocabList = vocabList.split("\n")
        vocabList_d = {}
        for ea in vocabList:
            [value, key] = ea.split("\t")
            vocabList_d[key] = value

        # 19
        processed_email = process_email("Content of email")
        print(processed_email)
        word_indices = find_word_indices(processed_email, vocabList_d)
        print(word_indices)

        # 20
        features = email_features(word_indices, vocabList_d)
        print(features.flatten())

        # 21
        email_sample1       =       open(os.path.join(os.path.dirname(__file__),       'data',
'emailSample1.txt'), "r").read()
        email_sample2       =       open(os.path.join(os.path.dirname(__file__),       'data',
'emailSample2.txt'), "r").read()
        spam_sample1 = open(os.path.join(os.path.dirname(__file__), 'data', 'spamSample1.txt'),
"r").read()
        spam_sample2 = open(os.path.join(os.path.dirname(__file__), 'data', 'spamSample2.txt'),
"r").read()

        email_sample1 = transform_email_to_features(email_sample1, vocabList_d)
        email_sample2 = transform_email_to_features(email_sample2, vocabList_d)
        spam_sample1 = transform_email_to_features(spam_sample1, vocabList_d)
        spam_sample2 = transform_email_to_features(spam_sample2, vocabList_d)

        print('Spam  -> 1\nEmail -> 0')
        print('Linear Kernel: ')
        print('False', classifier5.predict(email_sample1.T))
        print('False', classifier5.predict(email_sample2.T))
        print('Spam', classifier5.predict(spam_sample1.T))
        print('Spam', classifier5.predict(spam_sample2.T))
        print('\n Gaussian Kernel: ')
        print('False', classifier6.predict(email_sample1.T))
        print('False', classifier6.predict(email_sample2.T))
        print('Spam', classifier6.predict(spam_sample1.T))
        print('Spam', classifier6.predict(spam_sample2.T))
        print('\n')

        # 22
```

```
        spam_sample = "Hi, Kirill Zyusko, Items on your wishlist are now discounted! Some items
you've added to your personal wishlist are currently discounted on http://www.gog.com/ - you
can find all the details below or directly on your https://www.gog.com/account/wishlist."
        spam_sample = transform_email_to_features(spam_sample, vocabList_d)
        print('\n Gaussian Kernel: ')
        print('Spam', classifier6.predict(spam_sample.T))

        email_sample = "Hi Kirill Zyusko, Your job search status is open, but not actively
looking, but we need more information before accelerating your matches. Once you tell us just a
little bit more about yourself, we'll get you in front of companies and send you any new jobs
that match your interests."
        email_sample = transform_email_to_features(email_sample, vocabList_d)
        print('Email (should be false - at least gmail doesn\'t classify this message as a
spam)', classifier6.predict(email_sample.T))

        spam_sample3 = "Hi, Hope you are doing great. This is Josh from Avco Consulting Inc.
Inc is a global IT company based in Worcester, MA. Our leadership in the industry has been
established by our excellence in helping clients use Information Technology to achieve their
business objectives. Our core competencies are Information Technology (IT)services and Project
Management. I have available consultants with Excellent communication, analytical, and team work
skills. I would appreciate if you - or someone you can recommend share the suitable requirements
accordingly. Below is the list of the available consultant with different skills set and their
preferred locations."
        spam_sample3 = transform_email_to_features(spam_sample3, vocabList_d)
        print('Spam', classifier6.predict(spam_sample3.T))

        # 23 - 24
        # Look at utils/download.py

        # 25
        # Change #18 to vocab2.txt to see the difference


    if __name__ == '__main__':
        main()
```