Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:

«Машинное обучение»

По лабораторной работе №5

«Метод опорных векторов»

Выполнил: Сенькович Дмитрий Сергеевич

магистрант кафедры информатики

группы №858642

Проверил: Стержанов Максим Валерьевич

доцент, кандидат технических наук

Минск 2019

**Оглавление**

[1 Введение 3](#_Toc25766878)

[2 Линейная зависимость 5](#_Toc25766879)

[2.1 Постановка задачи 5](#_Toc25766880)

[2.2 Чтение данных 5](#_Toc25766881)

[2.3 График исходных данных 5](#_Toc25766882)

[2.4 Обучение SVM классификатора 6](#_Toc25766883)

[2.5 Различные коэффициенты параметра регуляризации 7](#_Toc25766884)

[2.6 Различные коэффициенты параметра регуляризации 9](#_Toc25766885)

[3 Нелинейная зависимость 12](#_Toc25766886)

[3.1 Постановка задачи 12](#_Toc25766887)

[3.2 Чтение данных 12](#_Toc25766888)

[3.3 График исходных данных 12](#_Toc25766889)

[3.4 Обработка данных с помощью Гауссовского ядра 13](#_Toc25766890)

[3.5 Обучение классификатора и визуализация разделяющей поверхности 14](#_Toc25766891)

[4 Подбор оптимальных параметров SVM 17](#_Toc25766892)

[4.1 Постановка задачи 17](#_Toc25766893)

[4.2 Чтение данных 17](#_Toc25766894)

[4.3 Графики исходных данных 17](#_Toc25766895)

[4.4 Поиск лучших параметров 19](#_Toc25766896)

[4.5 Визуализация лучшего решения 19](#_Toc25766897)

[5 Подбор оптимальных параметров SVM 21](#_Toc25766898)

[5.1 Постановка задачи 21](#_Toc25766899)

[5.2 Чтение данных 21](#_Toc25766900)

[5.3 Обучение модели и подбор лучших параметров 21](#_Toc25766901)

[5.4 Реализация функции предобработки текста 22](#_Toc25766902)

[5.5 Загрузка словаря 23](#_Toc25766903)

[5.6 Преобразование имейла в вектор 23](#_Toc25766904)

[5.7 Проверка работы классификатора на тестовых имейлах 24](#_Toc25766905)

[5.8 Проверка работы классификатора на своих имейлах 25](#_Toc25766906)

[6 Подбор оптимальных параметров SVM 28](#_Toc25766907)

[6.1 Постановка задачи 28](#_Toc25766908)

[6.2 Создание словаря 28](#_Toc25766909)

[6.3 Подготовление обучающей и валидационной выборки 32](#_Toc25766910)

[6.4 Подбор оптимальных параметров 34](#_Toc25766911)

[6.5 Проверим качество классификации 35](#_Toc25766912)

# 1 Введение

Метод опорных векторов - еще одна модель обучения с учителем в машинном обучение. Широко распространена на практике. Метод опорных векторов является лучшей альтернативой логистической регрессии, являясь более быстрым в вычислении, когда дело касается линейных моделей. Также, метод лучше предназначен для моделирования нелинейных зависимостей.

SVM пытается не только разделить данные, но и наибольший margin. Это позволяет SVM выдавать более предсказуемые результаты.

Первое отличие SVM от линейной регрессии: cost function. Выглядит она следующим образом:

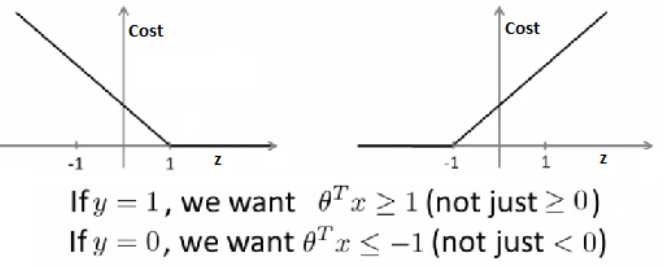


Рисунок 1.1. Функции частей стоимости SVM

Эти две функции апроксимируют функцию стоимости (каждая функция свою часть), что дает возможность для оптимизационных возможностей.

Функция стоимости в SVM выглядит так:

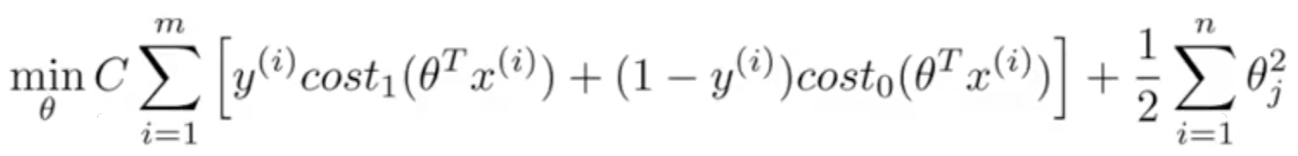


Рисунок 1.2. Функция стоимости SVM

Здесь параметр C играет ту же роль, что и параметр регуляризации (по смыслу 1/λ).

Далее, SVM в качестве выхода указывает не вероятность принадлежности к классу 0 или 1, а строго 0 или 1 по следующим критериям (следует из функции стоимости).

Различается и цель SVM от логистической регрессии. SVM стремится получить не просто достаточное условие минимизации функции стоимости, а чуть больше:

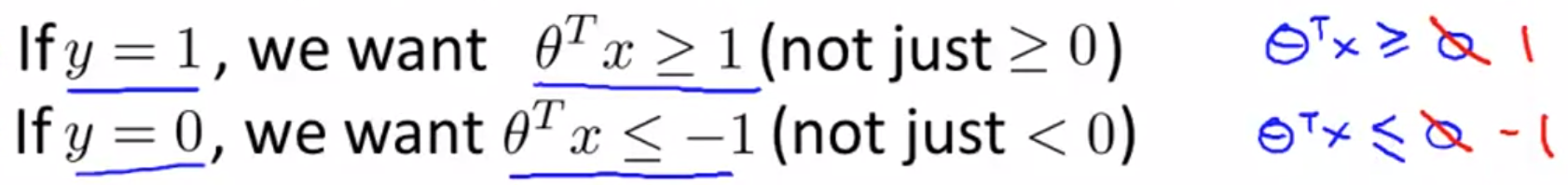
****

Рисунок 1.3. Задача SVM

Произведение выше представляет собой произведение проекции вектора x на длину вектора параметров θ. То есть, при достижении цели SVM, модель выберет лучшие минимальные значения вектора параметров, что позволит максимизировать длину проекций векторов выборки, в результате чего получится модель, разделяющая данные лучшим образом, менее подвергнутая аномальные значениям того или иного класса, почему SVM и может давать ответы лучше логистической регрессии:

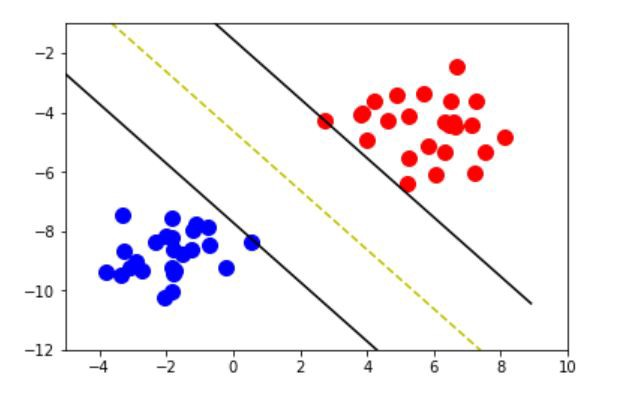


Рисунок 1.4. Decision boundary, которую может выбрать SVM в данной задаче

# 2 Линейная зависимость

### 2.1 Постановка задачи

Требуется разделить точки на плоскости, зависимость координат между которыми линейна.

### 2.2 Чтение данных

Считаем данные:

data = loadmat("ex5data1.mat")

x = data['X']

y = data['y']

### 2.3 График исходных данных

Построим данные, разделяя их разным цветом:

def plot\_data(x, y):

colors = ['green' if val else 'red' for val in y]

plt.scatter(x[:, 0], x[:, 1], c=colors)

plot\_data(x, y)

plt.show()

Полученный график:

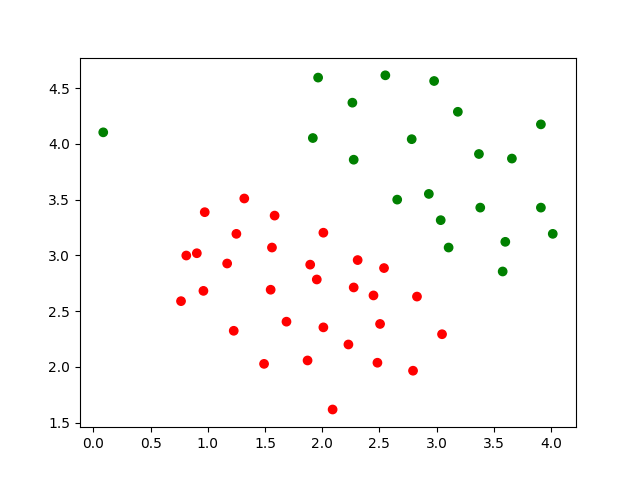


Рисунок 2.3.1. Исходные данные

### 2.4 Обучение SVM классификатора

Построим SVM модель с линейным ядром для классификации точек и построим график с decision boundary. Будем использовать библиотечную реализацию SVM из библиотеки scikit-learn из пакета sklearn.svm:

def plot\_linear\_decision\_boundary(svclassifier, x, y):

plot\_data(x, y)

w = svclassifier.coef\_[0]

a = -w[0] / w[1]

axx = np.linspace(-1, 5)

axy = a \* axx - (svclassifier.intercept\_[0]) / w[1]

plt.plot(axx, axy, 'k-')

plt.show()

svclassifier = SVC(kernel='linear')

svclassifier.fit(x, y.ravel())

plot\_linear\_decision\_boundary(svclassifier, x, y)

Получившийся график:

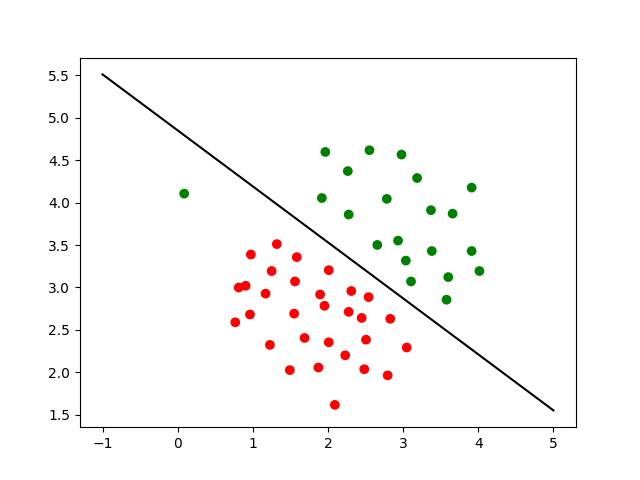


Рисунок 2.4.1. SVM с линейным ядром

### 2.5 Различные коэффициенты параметра регуляризации

Построим SVM c параметром регуляризации C = 1:

svclassifier = SVC(C = 1, kernel='linear')

svclassifier.fit(x, y.ravel())

plot\_linear\_decision\_boundary(svclassifier, x, y)

Получившийся график:

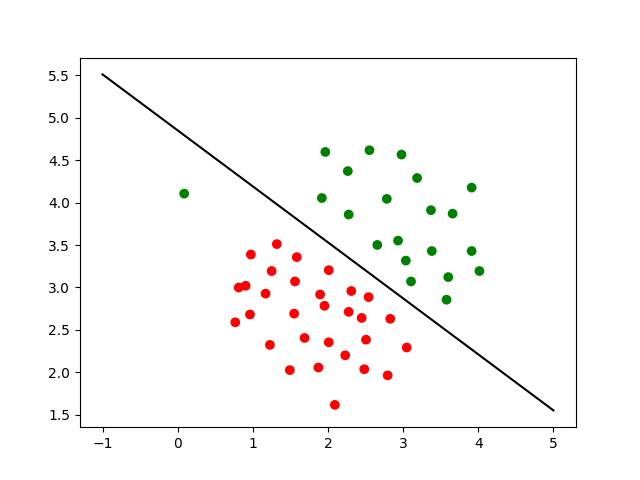


Рисунок 2.5.1. SVM с линейным ядром. Параметр регуляризации 1

Получили то же разделение, что и в предыдущем пункте, потому что C = 1 по умолчанию.

Построим SVM c параметром регуляризации C = 100:

svclassifier = SVC(C = 100, kernel='linear')

svclassifier.fit(x, y.ravel())

plot\_linear\_decision\_boundary(svclassifier, x, y)

Получившийся график:

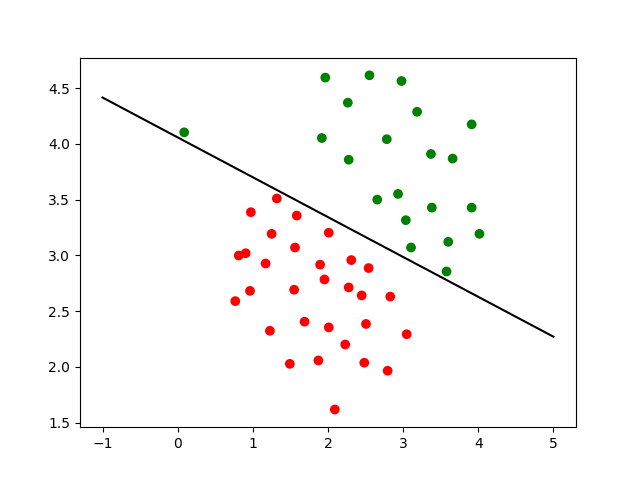


Рисунок 2.5.2. SVM с линейным ядром. Параметр регуляризации 100

Видим влияние параметра C, а именно overfitting. Объяснение этому следующее: увеличение параметра C соответствует уменьшению параметра λ в логистической регрессии, в результате чего уменьшается значение регуляризации, то есть модель больше подвержена проблеме переобучения, что мы и видим на графике выше.

### 2.6 Различные коэффициенты параметра регуляризации

Реализуем функцию вычисления Гауссового ядра для SVM:

def gaussian\_kernel(x1, x2, sigma):

distance\_length = np.sum(np.power(x1 - x2, 2))

return np.exp(-distance\_length / (2\*sigma\*\*2))

x1 = np.array([[1, 2, 1]])

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

sigma = 2

print(gaussian\_kernel(x1, x2, sigma))

Что такое ядра в SVM: это техника, позволяющая SVM строить сложные нелинейные модели.

Одно из возможных решений моделирования нелинейных отношений - добавление полиномиальных фич. Но таких фич может быть очень много, что в итоге сведется к невозможности использования алгоритма. По сути, добавление полиномиальных фич - это некоторый выбор фич. Поэтому целесообразно предположить, что можно подобрать такие фичи для конкретной задачи, которые позволят качественнее моделировать зависимость.

SVM дает ответ на вопрос подбора таких фич - ядра. Идея следующая: новыми фичами становятся степени близости оригинальных фич к некоторым другим ориентирам (landmarks). В нашем случае Гауссовского ядра этими ориентирами будут сами фичи, то есть первой новой фичей станут расстояния всех точек до нее, второй - до второй и т.д., а функция близости будет выглядеть следующим образом:

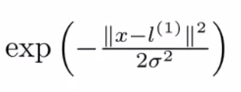


Рисунок 2.6.1. Функция близости Гауссовского ядра

Таким образом, чем ближе точка к заданном, тем выше значения близости (1 для самой фичи, так как евклидово расстояние будет 0, и до 0 в зависимости от дальности остальных фич).

В итоге получим некоторую нелинейную decision boundary, предсказывающую 1 исходя из меры экземпляра к тем или иными экземплярам обучающей выборки. В итоге функция стоимости преобразится в новую: фич станет столько, сколько экземпляров в обучающей выборке, а фичи будут использоваться новые.

Последнее, что стоит заметить в случае Гауссовского ядра: влияние параметра σ². В случае увеличения значения параметра фичи будут изменяться более плавно, то есть функция близости будет изменяться более медленно, что даст нам модель с более высоким bias. И, наоборот, в случае снижения параметра функция близость будет более “резкой”, формируя модель с более высокой variance:

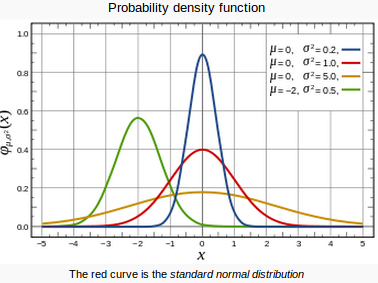


Рисунок 2.6.2. Различные значения параметра σ² и его влияние на функция близости

# 3 Нелинейная зависимость

### 3.1 Постановка задачи

Требуется разделить точки на плоскости, зависимость координат между которыми нелинейна.

### 3.2 Чтение данных

Считаем данные:

data = loadmat("ex5data2.mat")

x = data['X']

y = data['y']

### 3.3 График исходных данных

Построим данные, разделяя их разным цветом:

def plot\_data(x, y):

colors = ['green' if val else 'red' for val in y]

plt.scatter(x[:, 0], x[:, 1], c=colors)

plot\_data(x, y)

plt.show()

Полученный график:

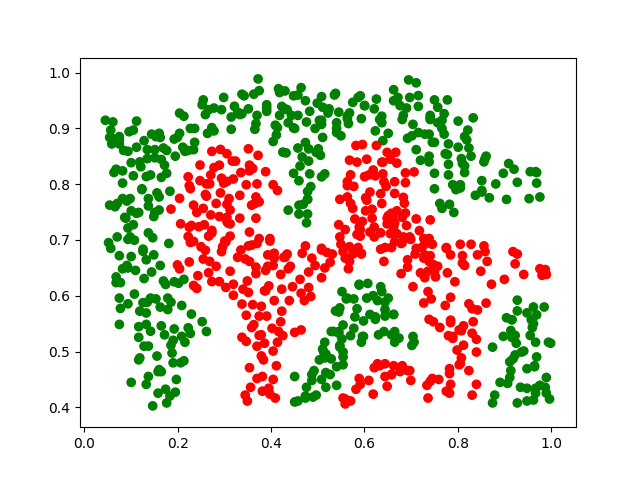


Рисунок 3.3.1. Исходные данные

### 3.4 Обработка данных с помощью Гауссовского ядра

Обработаем данные по логике выше:

def gaussian\_kernel\_single\_cell(x1, x2, sigma):

distance\_length = np.sum(np.power(x1 - x2, 2))

return np.exp(-distance\_length / (2\*sigma\*\*2))

def gaussian\_kernel(x, sigma):

m = x.shape[0]

kernel = np.zeros((m, m))

for i in range(m):

li = x[i]

for j in range(m):

xj = x[j]

kernel[j, i] = gaussian\_kernel\_single\_cell(xj, li, sigma)

return kernel

f = gaussian\_kernel(x, sigma)

### 3.5 Обучение классификатора и визуализация разделяющей поверхности

Обучим классификатор (C = 1, sigma = 0.1) и построим график разделяющей поверхности:

def plot\_nonlinear\_decision\_boundary(svclassifier, x, y, sigma):

x\_min, x\_max = x[:, 0].min(), x[:, 0].max()

y\_min, y\_max = x[:, 1].min(), x[:, 1].max()

ax = np.linspace(x\_min, x\_max, 100)

ay = np.linspace(y\_min, y\_max, 100)

xx = np.array([(axx, ayy) for axx in ax for ayy in ay])

xx\_in\_f = to\_f\_space(xx, x, sigma)

xx\_predictions = svclassifier.predict(xx\_in\_f)

colors = ['#90ee90' if prediction else '#ff9185' for prediction in xx\_predictions]

plt.scatter(xx[:, 0], xx[:, 1], c=colors, s = 3)

plot\_data(x, y)

plt.show()

sigma = 0.1

f = gaussian\_kernel(x, sigma)

svclassifier = SVC(C = 1, kernel = 'linear')

svclassifier.fit(f, y.ravel())

plot\_nonlinear\_decision\_boundary(svclassifier, x, y, sigma)

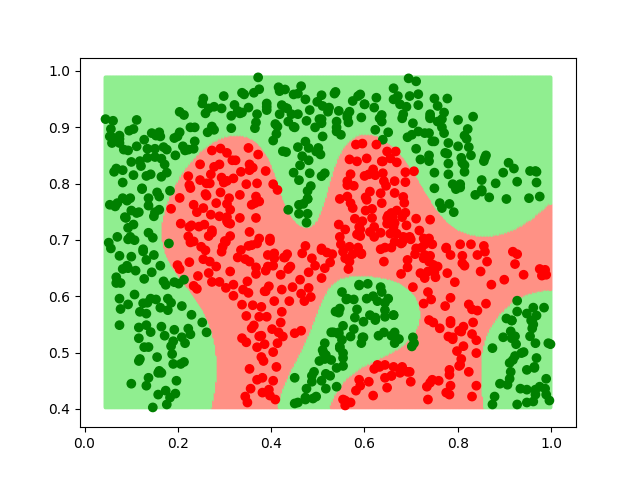


Рисунок 3.5.1. Разделяющая поверхность для SVM с Гауссовым ядром с параметрами C = 1, sigma = 0.1

Проделаем то же самое с параметром регуляризации 100. Получим следующий график:

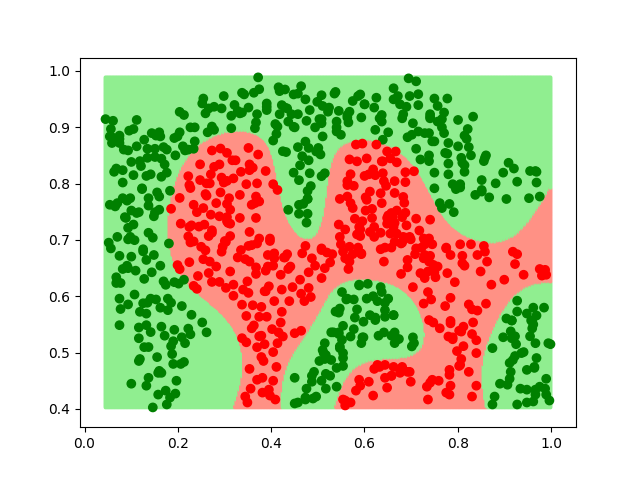


Рисунок 3.5.2. Разделяющая поверхность для SVM с Гауссовым ядром с параметрами C = 100, sigma = 0.1

Видим снова переобученную модель, так как данные разделены “идеально”.

# 4 Подбор оптимальных параметров SVM

### 4.1 Постановка задачи

Даны обучающая и валидационная выборки с двумя фичами. Требуется подобрать параметры модели SVM с Гауссовым ядром.

### 4.2 Чтение данных

Считаем данные:

data = loadmat("ex5data3.mat")

x = data['X']

y = data['y']

x\_val = data['Xval']

y\_val = data['yval']

### 4.3 Графики исходных данных

Построим данные из обучающей выборки, разделяя их разным цветом:

def plot\_data(x, y):

colors = ['green' if val else 'red' for val in y]

plt.scatter(x[:, 0], x[:, 1], c=colors)

plot\_data(x, x)

plt.show()

Полученный график:

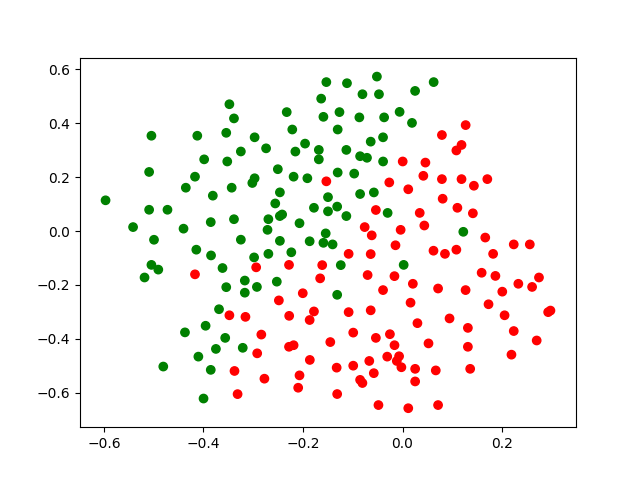


Рисунок 4.3.1. Данные из обучающей выборки

Теперь аналогично построим график данных из валидационной выборки:

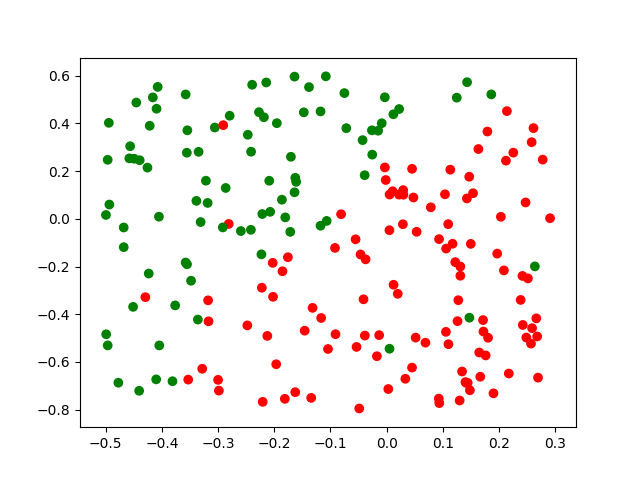


Рисунок 4.3.2. Данные из валидационной выборки

### 4.4 Поиск лучших параметров

Поиск параметров будем вести в интервале [0.1, 100] для параметра C (100 примеров) и в интервале [0.01, 1] для параметра γ (γ = 1/(2\*σ²); в таком виде библиотечная SVM использует параметра σ):

number\_of\_C\_samples = 100

number\_of\_gamma\_samples = 100

C\_to\_check = np.linspace(0.1, 100, number\_of\_C\_samples)

gamma\_to\_check = np.linspace(0.01, 1, number\_of\_gamma\_samples)

best\_C = -1

best\_sigma\_squared = -1

best\_score = -1

curr\_model\_number = 0

for C in C\_to\_check:

for gamma in gamma\_to\_check:

classifier = SVC(C=C, gamma=gamma, kernel='rbf')

classifier.fit(x, y.flatten())

score = classifier.score(x\_val, y\_val.flatten())

if score > best\_score:

best\_score = score

best\_C = C

best\_sigma\_squared = 1 / (2\*gamma)

curr\_model\_number += 1

if curr\_model\_number % 1000 == 0:

print(curr\_model\_number)

print('best\_C %s, best\_sigma\_squared %s, best\_score %s' % (best\_C, best\_sigma\_squared, best\_score))

Получим следующие значения лучшей модели: C = 1.1090909090909091 и σ² = 0.6329113924050632, что дает нам score 0.945.

### 4.5 Визуализация лучшего решения

Визуализируем модель на валидационной выборке:

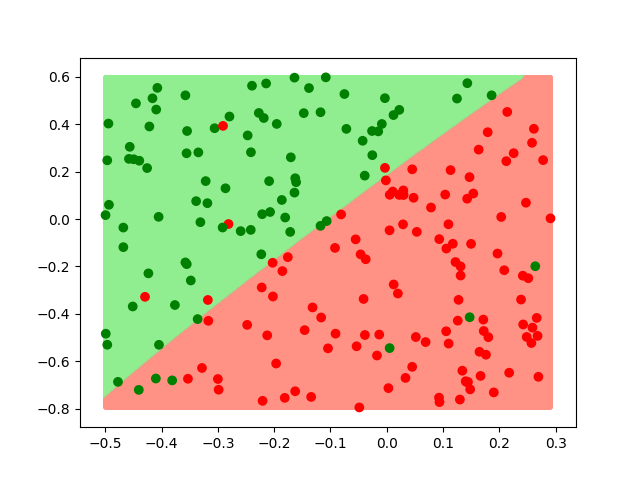


Рисунок 4.5.1. Модель с лучшими подобранными параметрами и валидационная выборка

Как видим, получилась довольно хорошая, стабильная модель, на которую не сильно влияют аномальные значения данных.

# 5 Подбор оптимальных параметров SVM

### 5.1 Постановка задачи

Требуется научить модель SVM отличать имейла двух видов: обычных и спам. Даны две выборки данных: обучающая и валидационная. В каждой выборке каждый экземпляр представляет из себя имейл, закодированный в следующем виде: каждая фича принимает значения 0 или 1, что соответствует наличию слова из некоторого словаря.

### 5.2 Чтение данных

Считаем данные:

data = loadmat("spamTrain.mat")

x = data['X']

y = data['y']

data\_test = loadmat("spamTest.mat")

x\_test = data\_test['Xtest']

y\_test = data\_test['ytest']

### 5.3 Обучение модели и подбор лучших параметров

Обучим модель и подберем оптимальные параметры аналогично предыдущему заданию:

number\_of\_C\_samples = 10

number\_of\_gamma\_samples = 100

C\_to\_check = np.linspace(1, 10, number\_of\_C\_samples)

gamma\_to\_check = np.linspace(0.01, 1, number\_of\_gamma\_samples)

best\_C = -1

best\_sigma\_squared = -1

best\_score = -1

curr\_model\_number = 0

for C in C\_to\_check:

for gamma in gamma\_to\_check:

classifier = SVC(C=C, gamma=gamma, kernel='rbf')

classifier.fit(x, y.flatten())

score = classifier.score(x\_test, y\_test.flatten())

if score > best\_score:

best\_score = score

best\_C = C

best\_sigma\_squared = 1 / (2\*gamma)

curr\_model\_number += 1

print(curr\_model\_number)

print(best\_C)

print(best\_sigma\_squared)

print(best\_score)

print('best\_C %s, best\_sigma\_squared %s, best\_score %s' % (best\_C, best\_sigma\_squared, best\_score))

Единственным различием будет другие интервалы для проверки параметров ([0.1; 1] и [0.01; 1] для C и γ соответственно) и количество C - 10 вместо 100.

Лучший score получится 0.987 при параметрах C = 1, σ² = 50.

### 5.4 Реализация функции предобработки текста

Реализуем функция очистки текста от html тегов, ссылок и т.д.:

special\_characters = "!@#$%^&\*()\_+=-?/{[}]/\*-+`~<>,.|\\:;\t\"\'―"

def remove\_html\_tags(text):

return re.sub('<.\*?>', ' ', text)

def replace\_urls(text):

return re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+]|[!\*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', ' httpaddr ', text)

def replace\_emails(text):

return re.sub('[^\s]+@[^\s]+', ' emailaddr ', text)

def replace\_numbers(text):

return re.sub('[0-9]+', ' number ', text)

def replace\_dollar\_signs(text):

return re.sub('[$]+', ' dollar ', text)

def remove\_special\_characters(text):

return ''.join(c for c in text if c not in special\_characters)

def stem(text):

stemmer = PorterStemmer()

text = [stemmer.stem(token) for token in text.split(" ")]

return " ".join(text)

def remove\_multiple\_spaces(text):

return re.sub(' +', ' ', text)

def clear\_email(email):

temp = email.lower()

temp = remove\_html\_tags(temp)

temp = replace\_urls(temp)

temp = replace\_emails(temp)

temp = replace\_numbers(temp)

temp = replace\_dollar\_signs(temp)

temp = remove\_special\_characters(temp)

temp = remove\_multiple\_spaces(temp)

cleared\_email = stem(temp)

return cleared\_email

Для стемминга слов (замены слова в некотором форме к наиболее вероятной общей форме) используется библиотека nltk.

### 5.5 Загрузка словаря

Загрузим словарь следующим образом:

vocabulary = {}

with open('vocab.txt', 'r') as vocabulary\_file:

for line in vocabulary\_file:

word\_mapping = line.split('\t')

vocabulary[word\_mapping[1].replace('\n', '')] = word\_mapping[0]

Каждому индексу фичи будет соответствовать слово, использующееся для кодировки фичи.

### 5.6 Преобразование имейла в вектор

Будем преобразовывать имейл в вектор фичей следующим образом: сначала будем очищать имейл с помощью функции выше, а затем будем формировать вектор длиной, равной количеству слов в словаре, кодирую каждую фичу в соответствии с присутствием слова в очищенной имейле: 1, если слово есть, и 0 - если нет:

def find\_words\_from\_vocabulary(text, vocabulary):

found\_words\_codes = set()

for word in text.split(' '):

if word in vocabulary:

found\_words\_codes.add(vocabulary[word])

return found\_words\_codes

def to\_feature\_vector(found\_words\_codes, vocabulary):

x = np.array([0]\*len(vocabulary))

for code in found\_words\_codes:

x[int(code) - 1] = 1

return x.reshape(1, -1)

def clear\_and\_map\_email\_to\_feature\_vector(email, vocabulary):

cleared\_email = clear\_email(email)

found\_words\_codes = find\_words\_from\_vocabulary(cleared\_email, vocabulary)

return to\_feature\_vector(found\_words\_codes, vocabulary)

### 5.7 Проверка работы классификатора на тестовых имейлах

Даны 4 имейла: два обычных и два спам. Проверим работу получившегося классификатора на них:

# ------------ read email samples ---------

email\_sample1 = ""

with open('emailSample1.txt', 'r') as file:

email\_sample1 = file.read().replace('\n', ' ')

email\_sample2 = ""

with open('emailSample2.txt', 'r') as file:

email\_sample2 = file.read().replace('\n', ' ')

spam\_sample1 = ""

with open('spamSample1.txt', 'r') as file:

spam\_sample1 = file.read().replace('\n', ' ')

spam\_sample2 = ""

with open('spamSample2.txt', 'r') as file:

spam\_sample2 = file.read().replace('\n', ' ')

# ------------ read email samples ---------

# ------------ clear and map emails to feature vectors ---------

email1 = clear\_and\_map\_email\_to\_feature\_vector(email\_sample1, vocabulary)

email2 = clear\_and\_map\_email\_to\_feature\_vector(email\_sample2, vocabulary)

spam1 = clear\_and\_map\_email\_to\_feature\_vector(spam\_sample1, vocabulary)

spam2 = clear\_and\_map\_email\_to\_feature\_vector(spam\_sample2, vocabulary)

# ------------ clear and map emails to feature vectors ---------

# ------------ classify sample emails ---------

print('is sample spam: %s' % svclassifier.predict(email1))

print('is sample spam: %s' % svclassifier.predict(email2))

print('is sample spam: %s' % svclassifier.predict(spam1))

print('is sample spam: %s' % svclassifier.predict(spam2))

# ------------ classify sample emails ---------

Наш классификатор способен распознать первые два имейла как не спам и вторые два как спам.

### 5.8 Проверка работы классификатора на своих имейлах

Добавим еще 4 имейла: также два не спам и два спам. Тексты имейлов следующие:

Re: Annotation based auditing fails for annotation properties underneath collection valued subpath

If you're running into the same issue, please have a look at DATACMNS-1609. Christoph Strobl created a PR for this problem, progress can be found there. Thanks for the PR and linking back to this issue!

----------------------------------------------------------------

Hey Dmitry,

Thanks for taking the time and sharing with me!

It sounds absolutely reasonable.

Let's stay in touch so that we can re-connect when conditions change.

I will follow you on LinkedIn so that our connection won't get lost.

P.S. Wish you luck with M.A. and personal stuff

----------------------------------------------------------------

Mark your calendar, ImageCon is back for 2020 and we’re inviting you to take the stage!

No need to be an unsung hero. Showcase your insights and hard work as you inspire your peers to create digital experiences that delight and win customers. ImageCon 2020 will be taking place April 22-23rd at the Fairmont Hotel in San Francisco, CA and our Call For Speakers is now open.

SUBMIT YOUR TALK

We are seeking passionate and knowledgeable speakers to share their stories, insights and expertise on topics spanning web performance, leveraging the Visual Web, creating greater engagement with media assets, and more.

Submit your talk abstract, bio and key takeaways by Wednesday, January 15th, 2020 at 11:59PT to be considered as a speaker for ImageCon 2020.

----------------------------------------------------------------

All Day DevOps 2019 is back online November 6th. We’re bringing 150 DevOps experts to you and your team free, online. Then with our Slack channel, you'll be able to ask questions and exchange ideas with 30,000 other DevOps practitioners at the conference.

Register today and learn from DevOps and industry leaders like:

Проверим работу классификатора на них:

# ------------ my own samples ---------

email\_sample1 = ""

with open('myEmailSample1.txt', 'r') as file:

email\_sample1 = file.read().replace('\n', ' ')

email\_sample2 = ""

with open('myEmailSample2.txt', 'r') as file:

email\_sample2 = file.read().replace('\n', ' ')

spam\_sample1 = ""

with open('mySpamSample1.txt', 'r') as file:

spam\_sample1 = file.read().replace('\n', ' ')

spam\_sample2 = ""

with open('mySpamSample2.txt', 'r') as file:

spam\_sample2 = file.read().replace('\n', ' ')

email1 = clear\_and\_map\_email\_to\_feature\_vector(email\_sample1, vocabulary)

email2 = clear\_and\_map\_email\_to\_feature\_vector(email\_sample2, vocabulary)

spam1 = clear\_and\_map\_email\_to\_feature\_vector(spam\_sample1, vocabulary)

spam2 = clear\_and\_map\_email\_to\_feature\_vector(spam\_sample2, vocabulary)

print('is sample spam: %s' % svclassifier.predict(email1))

print('is sample spam: %s' % svclassifier.predict(email2))

print('is sample spam: %s' % svclassifier.predict(spam1))

print('is sample spam: %s' % svclassifier.predict(spam2))

# ------------ my own samples ---------

Классификатор также способен распознать имейлы корректно.

# 6 Подбор оптимальных параметров SVM

### 6.1 Постановка задачи

Создать свой собственный словарь и обучить модель на нем.

### 6.2 Создание словаря

Для создания словаря будут использоваться данные из оригинального корпуса текстов - <http://spamassassin.apache.org/old/publiccorpus/>. Hams - обычные имейлы, spams - спам.

Формировать словарь будем следующим образом: считаем все имейлы, очистим их, разделим на слова и отберем 2000 самых распространенных слов:

special\_characters = "!@#$%^&\*()\_+=-?/{[}]/\*-+`~<>,.|\\:;\t\"\'―"

features\_count = 2000

def extract\_message\_from\_html(html):

text = re.sub('<head.\*?>.\*?</head>', '', html, flags=re.M | re.S | re.I)

text = re.sub('<a\s.\*?>', ' httpaddr ', text, flags=re.M | re.S | re.I)

text = re.sub('<.\*?>', '', text, flags=re.M | re.S)

text = re.sub(r'(\s\*\n)+', '\n', text, flags=re.M | re.S)

return unescape(text)

def extract\_message(email):

message = ""

message\_content\_type = ""

for part in email.walk():

part\_content\_type = part.get\_content\_type()

if part\_content\_type not in ("text/plain", "text/html"):

continue

message = str(part.get\_payload())

message\_content\_type = part\_content\_type

if part\_content\_type == "text/plain":

return message

if message\_content\_type == "text/html":

return extract\_message\_from\_html(message)

def remove\_html\_tags(text):

return re.sub('<.\*?>', ' ', text)

def replace\_urls(text):

return re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+]|[!\*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', ' httpaddr ', text)

def replace\_emails(text):

return re.sub('[^\s]+@[^\s]+', ' emailaddr ', text)

def replace\_numbers(text):

return re.sub('[0-9]+', ' number ', text)

def replace\_dollar\_signs(text):

return re.sub('[$]+', ' dollar ', text)

def remove\_special\_characters(text):

return ''.join(c for c in text if c not in special\_characters)

def stem(text):

stemmer = PorterStemmer()

text = [stemmer.stem(token) for token in text.split(" ")]

return " ".join(text)

def remove\_multiple\_spaces(text):

return re.sub(' +', ' ', text)

def replace\_new\_lines(text):

return text.replace('\n', ' ')

def clear\_email(email):

temp = email.lower()

temp = remove\_html\_tags(temp)

temp = replace\_urls(temp)

temp = replace\_emails(temp)

temp = replace\_numbers(temp)

temp = replace\_dollar\_signs(temp)

temp = remove\_special\_characters(temp)

temp = remove\_multiple\_spaces(temp)

temp = replace\_new\_lines(temp)

cleared\_email = stem(temp)

return cleared\_email

# ------------ read hams and spams ---------

hams = []

for file\_name in os.listdir("easy\_ham"):

if file\_name == 'cmds':

continue

with open('easy\_ham/%s' % file\_name, 'rb') as file:

msg = email.message\_from\_bytes(file.read())

hams.append(extract\_message(msg))

spams = []

for file\_name in os.listdir("spam"):

if file\_name == 'cmds':

continue

with open('spam/%s' % file\_name, 'rb') as file:

msg = email.message\_from\_bytes(file.read())

text\_from\_msg = extract\_message(msg)

if text\_from\_msg:

spams.append(text\_from\_msg)

print('read emails')

# ------------ read hams and spams ---------

# ------------ build vocabulary ---------

cleared\_hams = [clear\_email(ham) for ham in hams]

cleared\_spams = [clear\_email(spam) for spam in spams]

print('cleared emails')

ham\_words = []

for cleared\_ham in cleared\_hams:

ham\_words.extend([word for word in cleared\_ham.split(' ') if len(word) > 2])

print('extracted ham words')

spam\_words = []

for cleared\_spam in cleared\_spams:

spam\_words.extend([word for word in cleared\_spam.split(' ') if len(word) > 2])

print('extracted spam words')

all\_words = ham\_words + spam\_words

most\_common\_words = Counter(all\_words).most\_common(features\_count)

print('found most common words')

with open('vocab.txt', 'w') as file:

for i in range(len(most\_common\_words)):

file.write("%s\t%s\n" % (i, most\_common\_words[i][0]))

print('built vocabulary')

# ------------ build vocabulary ---------

### 6.3 Подготовление обучающей и валидационной выборки

После создания словаря мы можем отобразить наши имейлы в вектора фич. Для начала, определимся с размерами обучающей и валидационной выборками: пусть в обучающей выборке будет 80% экземпляров всех имейлов, а в валидационной 20%. Это будет соответствовать 2000 имейлов и 400 спам имейлов в обучающей выборке и 500 имейлов и 99 спам имейлов в валидационной (один спам не удалось прочитать). Итак, сконвертируем имейлы в вектора фич и сохраним результат:

# ------------ build training and validation samples ---------

shuffle(hams)

shuffle(spams)

hams\_train\_count = 2000

spams\_train\_count = 400

emails\_train\_count = hams\_train\_count + spams\_train\_count

x = np.zeros(shape=(emails\_train\_count, features\_count))

y = np.zeros(shape=(emails\_train\_count, 1))

for i in range(emails\_train\_count):

if i < hams\_train\_count:

ham\_feature\_vector = clear\_and\_map\_email\_to\_feature\_vector(hams[i], vocabulary)

x[i] = ham\_feature\_vector

y[i] = 0

else:

spam\_feature\_vector = clear\_and\_map\_email\_to\_feature\_vector(spams[i - hams\_train\_count], vocabulary)

x[i] = spam\_feature\_vector

y[i] = 1

print('built training set')

hams\_validation\_count = 500

spams\_validation\_count = 99

emails\_validation\_count = hams\_validation\_count + spams\_validation\_count

x\_val = np.zeros(shape=(emails\_validation\_count, features\_count))

y\_val = np.zeros(shape=(emails\_validation\_count, 1))

for i in range(emails\_validation\_count):

if i < hams\_validation\_count:

ham\_index = hams\_train\_count + i

ham\_feature\_vector = clear\_and\_map\_email\_to\_feature\_vector(hams[ham\_index], vocabulary)

x\_val[i] = ham\_feature\_vector

y\_val[i] = 0

else:

spam\_index = spams\_train\_count + i - hams\_validation\_count

spam\_feature\_vector = clear\_and\_map\_email\_to\_feature\_vector(spams[spam\_index], vocabulary)

x\_val[i] = spam\_feature\_vector

y\_val[i] = 1

print('built validation set')

savemat('spamTrain.mat', {'X': x, 'y': y})

savemat('spamTest.mat', {'Xtest': x\_val, 'ytest': y\_val})

print('saved train and validation samples')

# ------------ build training and validation samples ---------

### 6.4 Подбор оптимальных параметров

Загрузим получившиеся выборки и подберем оптимальные параметры модели:

# ------------ searching for the best C and sigma squared ---------

number\_of\_C\_samples = 5

number\_of\_gamma\_samples = 20

C\_to\_check = np.linspace(2, 4, number\_of\_C\_samples)

gamma\_to\_check = np.linspace(0.001, 0.02, number\_of\_gamma\_samples)

best\_C = -1

best\_sigma\_squared = -1

best\_score = -1

curr\_model\_number = 0

for C in C\_to\_check:

for gamma in gamma\_to\_check:

classifier = SVC(C=C, gamma=gamma, kernel='rbf')

classifier.fit(x, y.flatten())

score = classifier.score(x\_test, y\_test.flatten())

if score > best\_score:

best\_score = score

best\_C = C

best\_sigma\_squared = 1 / (2\*gamma)

curr\_model\_number += 1

print(curr\_model\_number)

print(best\_C)

print(best\_sigma\_squared)

print(best\_score)

print('best\_C %s, best\_sigma\_squared %s, best\_score %s' % (best\_C, best\_sigma\_squared, best\_score))

# best\_C 3.0, best\_sigma\_squared 62.5, best\_score 0.989983305509182

# ------------ searching for the best C and sigma squared ---------

Ввиду объема выборки, сократим количество проверяемых параметров C до 5 и γ до 20. Интервалы буду [2; 4] для C и [0.001; 0.02] для γ (интервалы найдены путем многократного подбора и уменьшения интервалов в ходе исследования лучшего score модели). В результате получим лучший C = 3, а лучшую σ² = 62.5. Лучший score, который удалось получить: 0.989983305509182.

### 6.5 Проверим качество классификации

Проверка предсказаний модели аналогична предыдущему заданию: загрузка имейла, отображение его текста в вектор фич и предсказание моделью. На имейлах лабораторной работы получилось отличить спам от не спам имейлов корректно, на моих собственных имейлах пришлось подбирать различное количество фич и параметров (на самом деле, подбор был намного шире, чем описан в лабораторной работе, в частности, были проверены значения порядка 100 значений количества фич от 100 до 10000 с различными натренированными модели). Более того, у модели есть трудности с распознаванием спам имейлов. Следовательно, модель сильно зависит от качества тренировочных данных и способа отображения имейлов в вектора фич, а также метода и деталей очистки самих имейлов.