Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра информатики

Лабораторная работа №5
«Метод опорных векторов»

Выполнил: Чёрный Родион Павлович
магистрант кафедры информатики
группа №858642

Проверил: доцент кафедры информатики
Стержанов Максим Валерьевич

Минск 2019

Постановка задачи

Набор данных **ex5data1.mat** представляет собой файл формата *.mat. Набор содержит три переменные X_1 и X_2 (независимые переменные) и y (метка класса). Данные являются линейно разделимыми.

Набор данных **ex5data2.mat** представляет собой файл формата *.mat. Набор содержит три переменные X_1 и X_2 (независимые переменные) и y (метка класса). Данные являются нелинейно разделимыми.

Набор данных **ex5data3.mat** представляет собой файл формата *.mat. Набор содержит три переменные X_1 и X_2 (независимые переменные) и y (метка класса). Данные разделены на две выборки: обучающая выборка, по которой определяются параметры модели; валидационная выборка, на которой настраивается коэффициент регуляризации и параметры Гауссового ядра.

Набор данных **spamTrain.mat** представляет собой файл формата *.mat . 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 . 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 X_1, X_2, 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 σ^2 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 σ^2.
17. Реализуйте функцию предобработки текста письма, включающую в себя:
    a. перевод в нижний регистр;
    b. удаление HTML тэгов;
    c. замена URL на одно слово (например, “httpaddr”);
    d. замена email-адресов на одно слово (например, “emailaddr”);
    e. замена чисел на одно слово (например, “number”);
    f. замена знаков доллара ($) на слово “dollar”;
    g. замена форм слов на исходное слово (например, слова “discount”, “discounts”, “discounted”, “discounting” должны быть заменены на слово “discount”). Такой подход называется stemming;
    h. 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?

Описание реализации

1. Загружаем данные из файла ex5data1.mat используя функцию loadmat из пакета scipy.io:

```
mat = scipy.io.loadmat('data/ex5data1.mat')
x = mat['X']
y = mat['y'].flatten()
```

2. График загруженного набора данных изображен на рисунке 1.

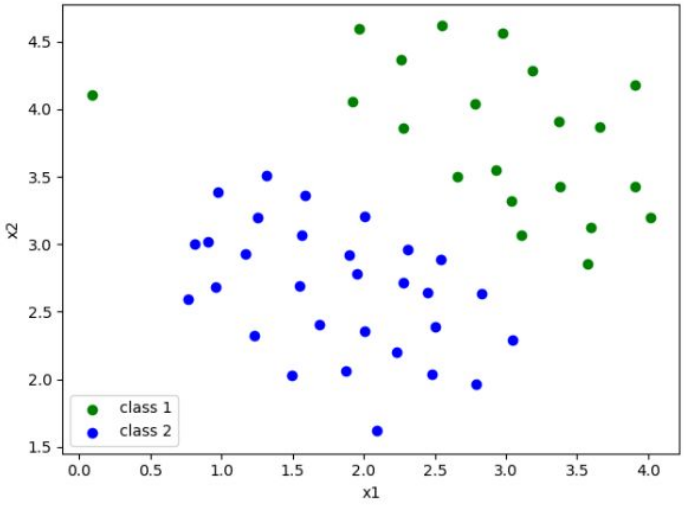

Рисунок 1 - Данные о принадлежности к классам из файла ex5data1.mat

3. Будем использовать реализацию алгоритма SVM из пакета sklearn.svm:

```
clf = svm.SVC(kernel='linear')
```

4. Обучим алгоритм SVM с параметрами C равными 1 и 100 и построим для каждого случая разделяющую прямую. Разделяющие прямые с исходными графиками изображены рисунках 2 и 3 соответственно. Чем больше параметр C, тем более точным для исходного набора данных будет прогноз. Можно увидеть, что в случае с C = 100 разделяющая правильно классифицирует отдаленную зеленую точку, которая, по правде говоря, скорее всего аномальная и не должна влиять на результат. И в случае с C=1 она как раз не влияет вид разделяющей прямой.

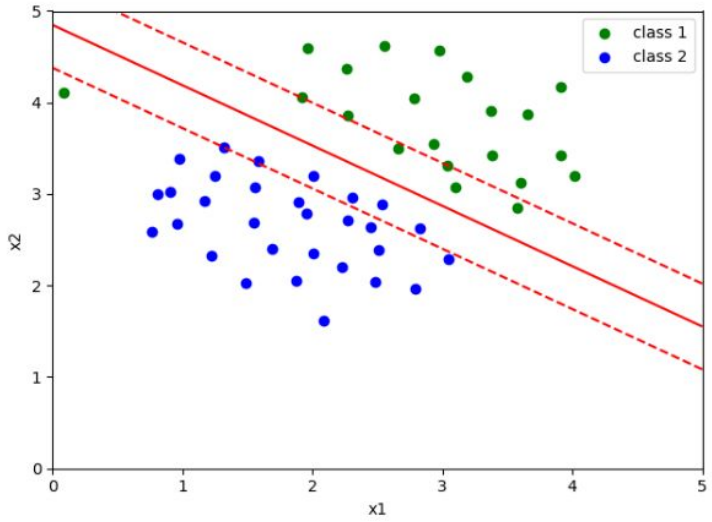

Рисунок 2 - Разделяющая прямая алгоритма SVM с C=1

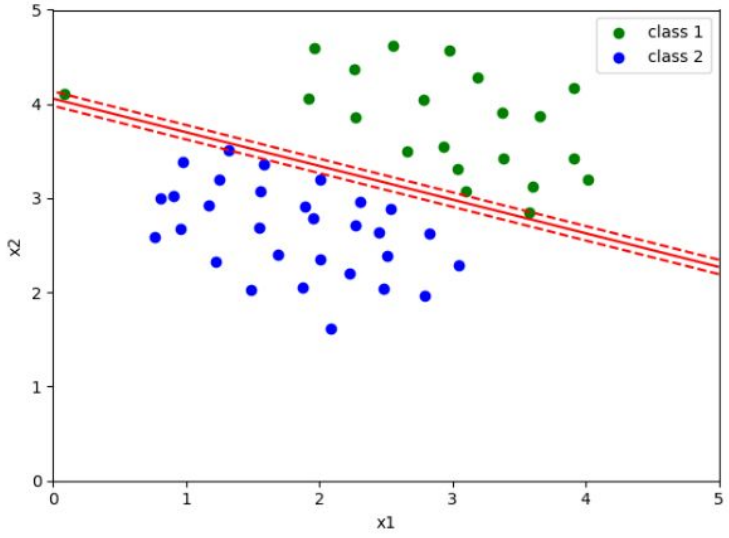

Рисунок 3 - Разделяющая прямая алгоритма SVM с C=100

5. Реализуем функцию вычисления Гауссовского ядра:

```
def gauss_kernel(x, l, gamma):
    matrix = np.power(x-l, 2)
    return np.exp(-np.sum(matrix, axis=1) * gamma)

def apply_kernel(x, gamma):
    return apply_kernel_with_points(x, x, gamma)

def apply_kernel_with_points(x, l, gamma):
    return np.array([gauss_kernel(x_i, l, gamma) for x_i in x])
```

6. Загрузим данные из файла ex5data2.mat:

```
mat = scipy.io.loadmat('data/ex5data2.mat')
x = mat['X']
y = mat['y'].flatten()
```

7. Обработаем исходные данные при помощи Гауссовского ядра. Положим sigma равным 1.

```
sigma = 1
gamma = 1 / (2 * np.power(sigma, 2))
f = apply_kernel(x, gamma)
```

8. Обучим классификатор на обработанных данных:

```
clf = svm.SVC(kernel='linear')
clf.fit(f, y)
```

9. Визуализация данных из файла ex5data2.mat вместе с разделяющей кривой обученного алгоритма изображена на рисунке 4.

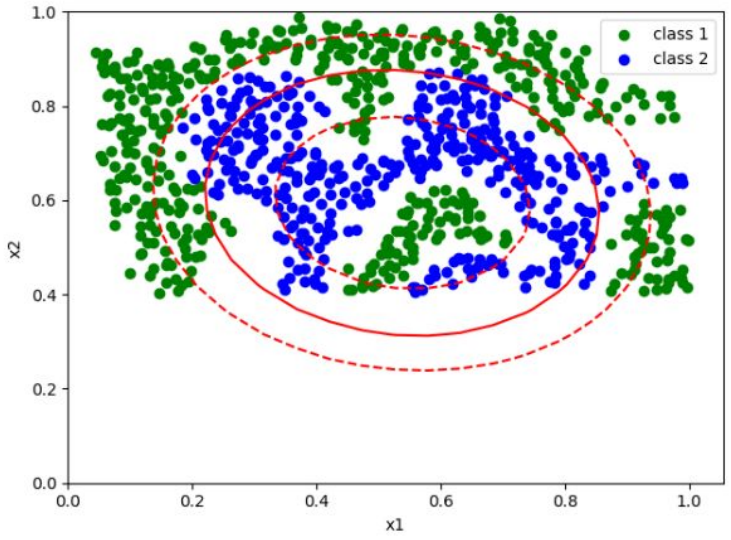

Рисунок 4 - данные из файла ex5data2.mat и разделяющая кривая

10. Загрузим данные для обучения и валидации из файла ex5data3.mat:

```
mat = scipy.io.loadmat('data/ex5data3.mat')
x = mat['X']
y = mat['y'].flatten()
x_val = mat['Xval']
y_val = mat['yval'].flatten()
```

11. Перебором определим параметры C и σ, при которых модель даст наилучшие результаты на валидационной выборке. Параметры C и σ будем перебирать из значений: 0.01, 0.03, 0.1, 0.3, 0.5, 1, 3, 10, 30, 50, 100.

Код для перебора и поиска лучшей пары параметров:

```
def find_better_params(x, y, x_val, y_val, c_values, sigma_values):
    c_optimal = 0
    sigma_optimal = 0
    result_score = 0
    for c in c_values:
        for s in sigma_values:
            gamma = np.power(s, -2.)
            classifier = svm.SVC(C=c, gamma=gamma, kernel='rbf')
            classifier.fit(x, y)
            score = classifier.score(x_val, y_val)
            print("C: ", c, "sigma: ", s, "score: ", score)
            if score > result_score:
                result_score = score
                c_optimal = c
                sigma_optimal = s

    return c_optimal, sigma_optimal
```

В результате перебора получаем, что наилучшими параметрами являются C= 0.3, σ = 0.1, при них, средняя точность предсказаний составляет 0.965.

12. Исходные данные вместе с разделяющей кривой изображены на рисунке 5.

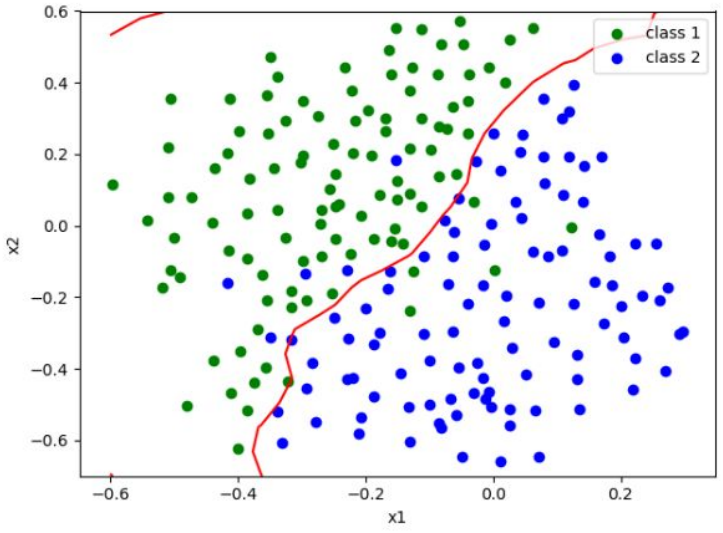

Рисунок 5 - данные из ex5data3.mat и разделяющая кривая модели с параметрами C= 0.3, σ = 0.1

13. Загрузим данные из файла spamTrain.mat:

```
mat = scipy.io.loadmat('data/spamTrain.mat')
x = mat['X']
y = mat['y'].flatten()
```

14. Будем обучать классификатор во время подбора параметров.

15. Загрузим данные из файла spamTest.mat:

```
mat = scipy.io.loadmat('data/spamTest.mat')
x_test = mat['Xtest']
y_test = mat['ytest'].flatten()
```

16. Перебором определим параметры C и σ, при которых модель даст наилучшие результаты на валидационной выборке. Параметры C и σ будем перебирать тех же значений, что и в пункте 11. Код перебора будем использовать тоже из пункта 11.

В результате получаем, что с параметрами C = 0.03 и σ = 0.3 точность классификации на тестовой сборке составляет 0.99, на тренировочной: 0.99425

17. Реализуем функции предобработки текста. Для стемминга будем использовать тип PorterStemmer из пакета nltk.stem. А для определения url - типа URLExtract из пакета urlextract.

```
class TextPreprocessor:
    def __init__(self):
        self.stemmer = PorterStemmer()
        self.url_extractor = urlextract.URLExtract()
        self.tag_regex = re.compile(r"<[^>]*>")
        self.email_regex = re.compile(r"[^\s]+@[^\s]+")
        self.number_regex = re.compile(r'\d+(?:\.\d*(?:[eE]\d+))?')
        self.dollar_regex = re.compile(r"[$]+")
        self.spaces_regex = re.compile(r"\s+")
        self.special_chars = [
            "<", "[", "^", ">", "+", "?", "!", "'", ".", ",", ":",
            "*", "%", "#", "_", "=", "-", "&", '/', '\\', '(', ')'
        ]
    def preprocess_text(self, text):
        text = text.lower()
        text = self.remove_html_tags(text)
        text = self.replace_urls(text)
        text = self.replace_emails(text)
        text = self.replace_numbers(text)
        text = self.replace_dollar_signs(text)
        text = self.stem_words(text)
```

```
        text = self.remove_special_characters(text)
        text = self.spaces_regex.sub(' ', text)
        return text.strip()
    def remove_html_tags(self, text):
        text = self.tag_regex.sub(" ", text).split(" ")
        text = filter(len, text)
        text = ' '.join(text)
        return text
    def replace_urls(self, text):
        urls = list(set(self.url_extractor.find_urls(text)))
        urls.sort(key=lambda u: len(u), reverse=True)
        for url in urls:
            text = text.replace(url, " httpaddr ")
        return text
    def replace_emails(self, text):
        return self.email_regex.sub(" emailaddr ", text)
    def replace_numbers(self, text):
        return self.number_regex.sub(" number ", text)
    def replace_dollar_signs(self, text):
        return self.dollar_regex.sub(" dollar ", text)
    def remove_special_characters(self, text):
        for char in self.special_chars:
            text = text.replace(str(char), "")
        text = text.replace("\n", " ")
        return text
    def stem_words(self, text):
        text = [self.stemmer.stem(token) for token in text.split(" ")]
        text = " ".join(text)
        return text
```

18. Загрузим слова и их коды из словаря vocab.txt. Если слово не присутствует в словаре, то мы его помечаем кодом 0, с соответствующим токеном: unknown.

```
word_to_code = {}
code_to_word = ["unknown"]
with open("data/vocab.txt") as f:
    for line in f:
        code, word = line.split('\t')
        word = word.strip()
        code = int(code)
        code_to_word.append(word)
        word_to_code[word] = code
```

19, 20. Реализуем функцию, которая будет превращать текст в список векторов-признаков. Для этого мы сначала делаем препроцессинг текста с помощью класса из пункта 17, затем переводим каждое слово в код с помощью отображения, которое мы загрузили из словаря и устанавливаем соответствующие индексы нулевого вектора в единицу.

```
def convert_text_to_feature_vector(text, mapping):
    preprocessor = TextPreprocessor()
    text = preprocessor.preprocess_text(text)
    words = text.split(' ')
    feature_vector = np.zeros(len(mapping))
    for word in words:
        if word in mapping:
            feature_vector[mapping[word] - 1] = 1
    return feature_vector
```

21. Проверим работу классификатора на примерах из файлов emailSample1.txt, emailSample2.txt, spamSample1.txt и spamSample2.txt. Результат работы классификатора:

data/emailSample1.txt is not a spam
data/emailSample2.txt is not a spam
data/spamSample1.txt is a spam
data/spamSample2.txt is a spam

22, 23, 24 Логика для подготовки собственного словаря, а также формирования обучающих и тестировочных наборов данных находится в файле prepare_corpus.py. В самом модуле prepare_corpus мы проводим следующие действия:

- скачиваем архивы писем, являющихся спамом (20030228_spam.tar.bz2) и архивы писем, которые спамом не являются (20030228_easy_ham.tar.bz2), из корпуса http://spamassassin.apache.org/old/publiccorpus.
- Разархивируем письма соответственно в папки ham и spam
- К каждому спам-письму применяем функцию предобработки текста из пункта 17, разбиваем текст на слова, убираем стоп-слова и добавляем каждое слово в словарь-счетчик
- Потом сортируем слова в счетчике в порядке убывания их количества и берем 1899 самых часто встречающихся слов.
- Сохраняем каждое слово под уникальным индексом в файл vocab2.txt.
- Из 80% спам-писем и обычных писем формируем обучающий набор данных. Из оставшихся 20% формируем тестировочный набор. Сохраняем в файлы customTrain.mat и customTest.mat соответственно.

В результате у нас на выходе будет 3 файла:

- vocab2.txt - словарь, сформированный из множества электронных писем,
- customTrain.mat, custom.Test.mat - данные для обучения модели и ее тестирования, разделенные из исходного набора данных в отношении 80% / 20%.

25. Проведем обучение и тестирование модели на новом наборе данных. А также проверим работу классификатора на тех же примерах emailSample1.txt, emailSample2.txt, spamSample1.txt и spamSample2.txt.

Результат работы классификатора:

Accuracy on custom training set: 98.375 %
Accuracy on custom test set: 98.33610648918469 %

data/emailSample1.txt is not a spam
data/emailSample2.txt is not a spam
data/spamSample1.txt is a spam
data/spamSample2.txt is a spam