Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №5
«Метод опорных векторов»

БГУИР 1-40 81 04

Магистрант:
гр. 858642
Кукареко А.В.

Проверил:
Стержанов М. В.

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:
    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?
26. Ответы на вопросы представьте в виде отчета.

Результат выполнения:

1. Загрузите данные ex5data1.mat из файла..

```
data = scipy.io.loadmat('ex5data1.mat')
```

```
X = data['X']
y = data['y']
```

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

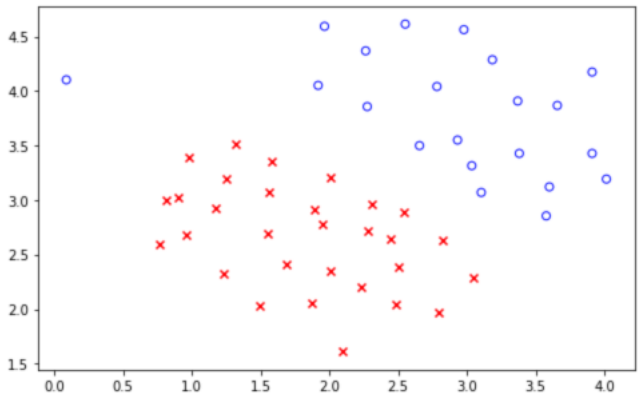

Рисунок 1 – гарфик данных из набора ex5data1.mat.

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

```
linear_svm_c1 = svm.SVC(C=1, kernel='linear')
linear_svm_c1.fit( X, y.flatten() )
```

```
SVC(C=1, 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?

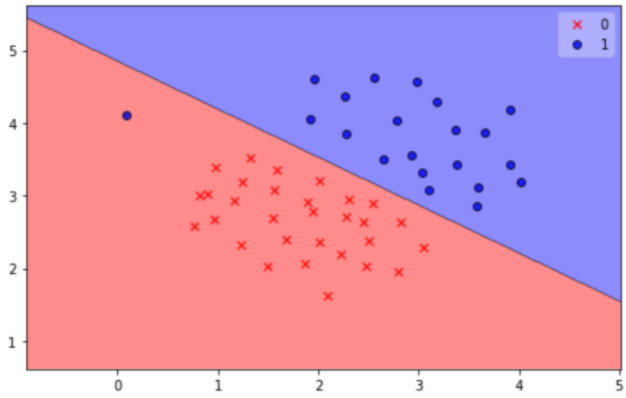

Рисунок 2 – график разделяющей прямой при C = 1.

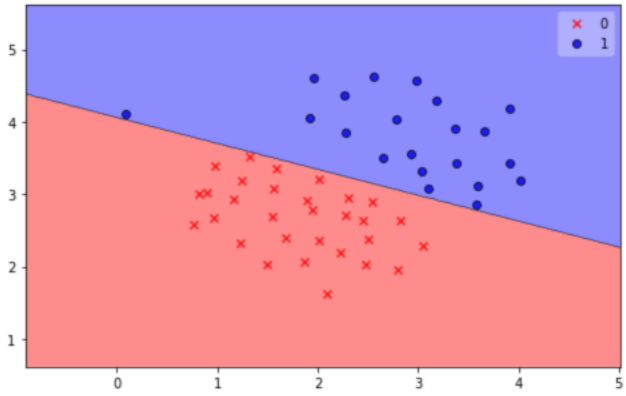

Рисунок 3 – график разделяющей прямой при C = 100.

При C = 1 алгоритм попытался поулчить наибольший "margin" между классами, при этом некоторые "выбросы" были классифицированы ошибочно, но общий паттерн алгоритм определил. При C = 100 видно, что верно были классифицированы все данные, но "margin" сильно уменьшился. Видно, что модель страдает от переобучения (overfitting high variance)

5. Реализуйте функцию вычисления Гауссового ядра для алгоритма SVM.

Функция вычисления Гауссового ядра:

```
def gaussian_kernel_similarity(x1, l1, sigma):
    diff = x1 - l1

    numerator = np.dot(diff.T, diff) # нужно взять корень квадратный а потом возьвести в 2 степень

    denominator = (2 * (sigma ** 2))

    return np.exp(-numerator/ denominator)
```

Функция генерации векторов фич f:

```
def create_gaussian_f(X, L, sigma):
    xm = len(X)
    lm = len(L)
    fs = np.zeros([xm, lm])
    for xi in range(xm):
        x = X[xi]
        for lj in range(lm):
            l = L[lj]
            fs[xi][lj] = gaussian_kernel_similarity(x, l, sigma)

    return fs
```

6. Загрузите данные ex5data2.mat из файла.

```
data2 = scipy.io.loadmat('ex5data2.mat')
```

```
X2 = data2['X']
y2 = data2['y']
```

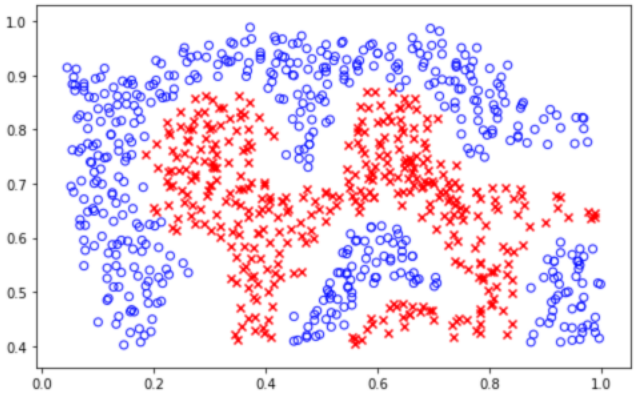

Рисунок 4 – гарфик данных из набора ex5data2.mat.

7. Обработайте данные с помощью функции Гауссового ядра.

```
sigma_ex2 = 0.1
```

```
F = create_gaussian_f(X2, X2, sigma_ex2)
F.shape
```

```
(863, 863)
```

8. Обучите классификатор SVM.

Обучение линейного классификатора фичам f:

```
svm_ex_2 = svm.SVC(C=1, kernel='linear')
svm_ex_2.fit( F, y2.flatten() )
```

Альтернативный вариант, использующий ядро библиотеки sklearn «rbf»:

```
def sigma_to_gamma_ang(sigma):
    return np.power(sigma, -2.) / 2

sigma_to_gamma_ang(2)
```

```
0.125
```

```
svm_ex_2_alt = svm.SVC(C=1, kernel='rbf', gamma=sigma_to_gamma_ang(sigma_ex2))
svm_ex_2_alt.fit( X2, y2.flatten() )
```

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

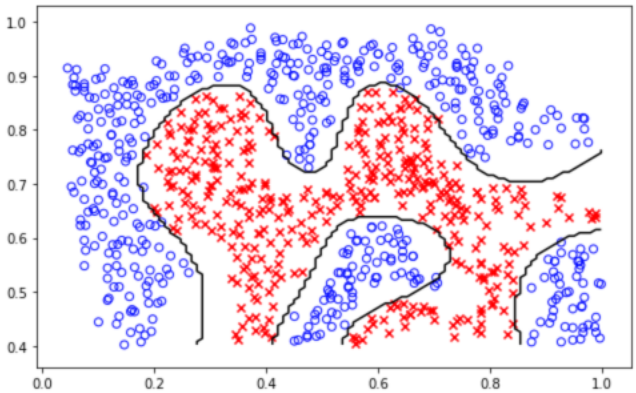

Рисунок 5 – визуализация разделяющий кривой, с использованием данных преобразованных собственной реализацией Гауссового ядра.

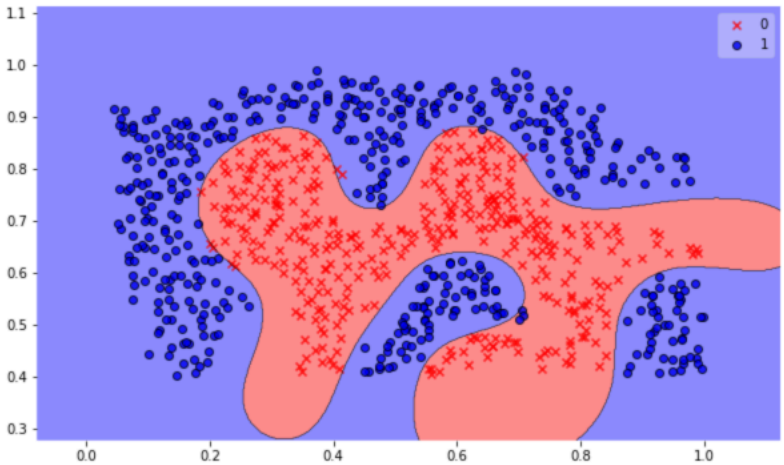

Рисунок 6 – визуализация разделяющий кривой, с использованием ядра библиотеки sklearn «rbf».

10. Загрузите данные ex5data3.mat из файла.

```
data3 = scipy.io.loadmat('ex5data3.mat')
```

```
X3_train = data3['X']
y3_train = data3['y']
X3_val = data3['Xval']
y3_val = data3['yval']
```

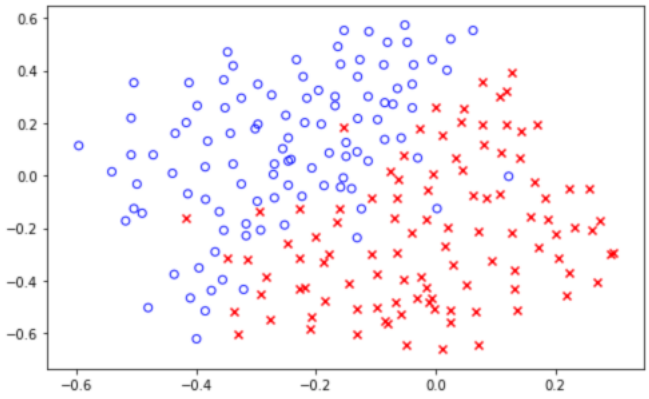

Рисунок 7 – гарфик данных из набора ex5data3.mat.

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

Во время эксперимента использовались следующие данные:

```
C_vals = [0.01, 0.04, 0.08, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5, 10, 20, 40, 80, 100]
sigma_vals = [0.01, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.4, 0.8, 1, 2, 5, 10, 20, 40, 80, 100]
```

Алгоритм вычисления лучших параметров:

```
for C in C_vals:
    for sigma in sigma_vals:
        model = svm.SVC(C=C, kernel='rbf', gamma=sigma_to_gamma_ang(sigma))
        model.fit(X3_train, y3_train.flatten())
        score = model.score(X3_val, y3_val)
        if score > best_score:
            best_score = score
            params['C'] = C
            params['sigma'] = sigma
```

Лучшая комбинация набрала 0.97, при C = 0.3, sigma = 0.08

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

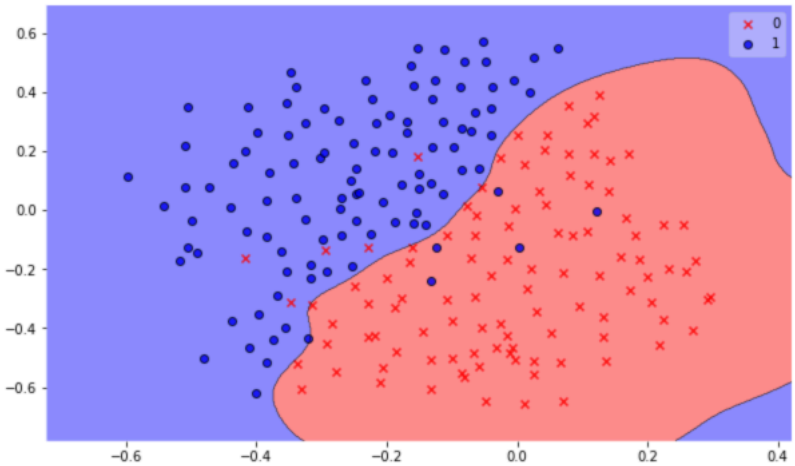

Рисунок 8 – визуализация разделяющий кривой.

13. Загрузите данные spamTrain.mat из файла.

```
data1 = scipy.io.loadmat('spamTrain.mat')
X1 = data1['X']
y1 = data1['y']

X1.shape
```

```
(4000, 1899)
```

14. Обучите классификатор SVM.

```
model1 = svm.SVC(C=0.5, kernel='linear')
model1.fit( X1, y1.flatten() )
print('spamTrain accuracy: ', (model1.score(X1, y1.flatten())) * 100)
```

```
spamTrain accuracy:  99.97500000000001
```

Точность на обучающей выборке получилась 99.97%

15. Загрузите данные spamTest.mat из файла.

```
data2 = scipy.io.loadmat('spamTest.mat')
X1_test = data2['Xtest']
y1_test = data2['ytest']
```

```
print('spamTest accuracy: ', (model1.score(X1_test, y1_test.flatten())) * 100)
```

```
spamTest accuracy:  97.6
```

Точность на тестовой выборке получилась 97.6%

16. Подберите параметры C и σ2.

Подбор осуществлялся следующими параметрами:

```
C_vals = [0.01, 0.04, 0.08, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5, 10, 20, 40, 80, 100]
sigma_vals = [0.01, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.4, 0.8, 1, 2, 5, 10, 20, 40, 80, 100]
```

Результат для ядра Гаусса: C=5, σ = 10.
Результат для линейного ядра: C=0.04.

17. Реализуйте функцию предобработки текста письма

Функция предобработки текста письма:

```
def prepare_text(text):
    import re

    text = text.lower() #перевод в нижний регистр;
    text = re.sub('<a .*href="([^"]*)".*?<\/a>', r' \1 ', text) # до
    text = re.sub('<[^<>]+>', ' ', text) #удаление HTML тэгов
    text = re.sub('(http|https)://[^\s]*', ' httpaddr ', text) #замен
    text = re.sub('[^\s]+@[^\s]+', ' emailaddr ', text) #замена email
    text = re.sub('[0-9]+', ' number ', text)      #замена чисел на одн
    text = re.sub('[$]+', ' dollar ', text)      #замена знаков доллара
    #остальные символы должны быть удалены и заменены на пробелы, т.е. в р
    text = re.sub('[_]+', ' ', text)
    text = re.sub('[^\w]', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip()
    #замена форм слов на исходное слово (например, слова "discount", "disc
    stemmer = nltk.stem.porter.PorterStemmer()
    words = text.split()
    stemmed_words = []
    for word in words:
        stemmed = stemmer.stem(word)
        if not len(stemmed): continue
        stemmed_words.append(stemmed)

    return ' '.join(stemmed_words)
```

18. Загрузите коды слов из словаря vocab.txt.

Функция загрузки словаря:

```
def load_vocab(filename):
    dict = {}
    with open(filename) as f:
        for line in f:
            (val, key) = line.split()
            dict[key] = int(val)
    return dict
```

```
vocab = load_vocab('vocab.txt')
```

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

Функция преобразования текста в вектор:

```
def text_to_vec(text, vocab):
    n = len(vocab.keys())
    vec = np.zeros(n)

    words = prepare_text(text).split()
    for word in words:
        if word not in vocab: continue
        idx = vocab[word]
        vec[idx] = 1

    return vec.reshape(1, -1)
```

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

Функция преобразования текста файла в вектор:

```
def file_to_vec(filename, vocab):
    text = open(filename,'r').read()

    return text_to_vec(text, vocab)
```

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

Результат работы классификатора с линейным ядром:

```
print('Predict with Linear kernel')
predict_files(best_linear_model, emails_data)
```

```
Predict with Linear kernel
File: emailSample1.txt - 0. Predicted: [0]
File: emailSample2.txt - 0. Predicted: [0]
File: spamSample1.txt - 1. Predicted: [1]
File: spamSample2.txt - 1. Predicted: [1]
---------------------------
Accurecy: 100.0%
```

Результат работы классификатора с ядром Гаусса:

```
print('Predict with Gaussian kernel')
predict_files(best_gaussian_model, emails_data)
```

```
Predict with Gaussian kernel
File: emailSample1.txt - 0. Predicted: [0]
File: emailSample2.txt - 0. Predicted: [0]
File: spamSample1.txt - 1. Predicted: [1]
File: spamSample2.txt - 1. Predicted: [1]
---------------------------
Accurecy: 100.0%
```

На таком маленьком датасете, классификатор показал такую-же точность, как и классификатор с линейным ядром.

22. Также можете проверить его работу на собственных примерах.

В качестве «собственного» примера я использовал датасет из Kaggle, по ссылке https://www.kaggle.com/karthickveerakumar/spam-filter

Датасет состоит из 2900 записей, из них 1368 записи - спам, и 1532 - не спам.

Результат работы классификатора с линейным ядром:

```
print('Predict with Linear kernel')
predict_text(best_linear_model, kaggle_emails)
```

```
Predict with Linear kernel
---------------------------
Accurecy: 71.78999999999999%
```

Результат работы классификатора с ядром Гаусса:

```
print('Predict with Gaussian kernel')
predict_text(best_gaussian_model, kaggle_emails)
```

```
Predict with Gaussian kernel
---------------------------
Accurecy: 71.86%
```

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

Для своего набора данных я использовал 3 файла:

- 20050311_spam_2.tar.bz2;
- 20030228_easy_ham.tar.bz2;
- 20030228_spam.tar.bz2.

После распаковки, у меня получилось: 1897 – спам письма и 2501 – не спам письма. Для обработки писем и вытягивания из них «тела», была использована библиотека «email».

24. Постройте собственный словарь.

Для построения собственного словаря я каждое письмо прогнал через функцию очистки из задания № 17, затем подсчитал повторение слов, и выбрал 1899 самых частых для словаря. Мой словарь отличается от словаря из задания № 18.

25. Как изменилось качество классификации? Почему?

Классификаторы использующие мой словарь и мой датасет выдают результат чуть-чуть хуже, чем классификатор построенный на данных из задания.

Для файлов из задания № 21 линейный и Гаусса выдают по 75%:

```
print('Predict with Linear kernel')
predict_files(my_linear_model, emails_data, my_vocab)
```

```
Predict with Linear kernel
---------------------------
Accurecy: 75.0%
```

```
print('Predict with gaussian kernel')
predict_files(my_gaussian_model, emails_data, my_vocab)
```

```
Predict with gaussian kernel
---------------------------
Accurecy: 75.0%
```

На моих примерах точность упала в среднем на 2%:

```
print('Predict with Linear kernel')
predict_text2(my_linear_model, kaggle_emails, my_vocab)
```

```
Predict with Linear kernel
---------------------------
Accurecy: 70.28%
```

Результат классификатора с линейным ядром – 70.28% против 71.79%.

```
print('Predict with gaussian kernel')
predict_text2(my_gaussian_model, kaggle_emails, my_vocab)
```

```
Predict with gaussian kernel
----------------------------
Accurecy: 69.66%
```

Результат классификатора с ядром Гаусса – 69.66% против 71.86%.

Улучшить показатели классификатора можно следующими спомобами:

- Увеличить кол-во тренировочных экземпляров, я использовал только часть текстов из корпуса «spamassassin»;
- Более качественно обрабатывать письма, например включать заголовки, емейлы отправителя, подписи, итд.
- Проанализировать словарь и влияние каждого слова на определение спам/не спам. Составить «блек-лист» слов, которые часто используются но имеют очень маленький вес.

Вывод.

В ходе лабораторной работы я изучил принципы работы алгоритма «Support Vector Machine», а так же его составляющих: линейного ядра и ядра Гаусса. Мной были получены практические навыки обработки данных, а именно текста для составления словаря. Так же при реализации спам фильтра я получил опыт применения машинного обучения для задачи классификации текста.