НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:
«Лабораторная работа №6»

Выполнил: Осипов Лев,
студент группы 301ПИ (1).

Москва, 2015 г.

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Задание 1

Разделяющая поверхность при проекции на исходное пространство будет иметь форму эллипса, дуги эллипса или же прямой. Представим, что объекты разных классов в пространстве расположены на двух разных дугах одного эллипса и равноудалены друг от друга. В таком случае проекция разделяющей плоскости будет дугой, которая проходит между объектами классов.

Задание 2

1. Линейный SVM

У нас есть точки в d пространствах и необходимо разделить их плоскостью. После обучения мы имеем разделяющую гиперплоскость в d пространствах, описываемую d элементами. Каждый объект так же описывается d признаками, а для его классификации нам необходимо вычислить его местоположение относительно гиперплоскости. Так как количество признаков d, нам потребуется **O(d)** операций для классификации.

2. Ядерный SVM

В случае же ядерного SVM некоторые ядра могут иметь размерности, стремящиеся к бесконечности. Можно пойти по-другому: так как алгоритм использует только опорные вектора (формула линейного классификатора из презентации), возьмем их количество за N. Также присутствует зависимость от некой функции ядра сложностью K. Поэтому итоговая сложность будет **O(N)*O(K)**.

Задание 3

Сложение ядер происходит на основе конкатенации векторов. Поэтому мы сможем линейно разделить выборку, если проведем плоскость в пространстве первого ядра с учетом конкатенации векторов обоих ядер.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Для решения задания была написана программа, исследующая зависимость работы алгоритма от решающих правил для линейного и RBF ядер.

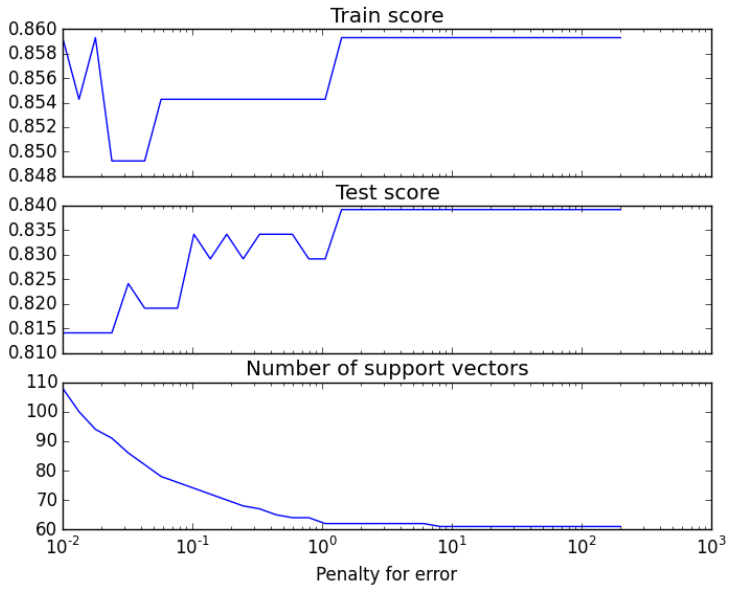

Рис. 1. Зависимость верно классифицированных объектов на обучающей и тестовой выборках, а также количество опорных векторов от C. Линейное ядро

Рис. 2. Зависимость верно классифицированных объектов на обучающей и тестовой выборках, а также количество опорных векторов от C. RBF ядро

В обоих ядрах наблюдается снижение количества опорных векторов, однако в случае RBF ядра снижение происходит медленнее.

Также в обоих ядрах наблюдается возрастание точности классификации, однако в случае RBF ядра на обучающей и тестовой выборках динамика возрастания весьма схожа, в отличие от случая линейного ядра.

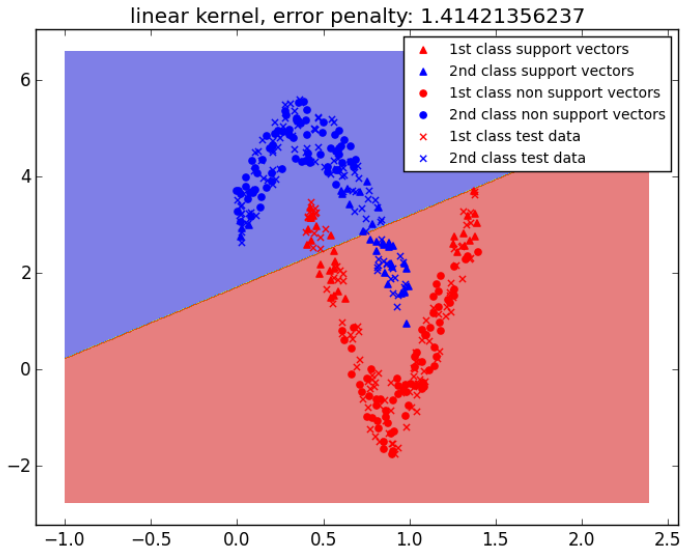

Рис. 3. Оптимальное C. Линейное ядро

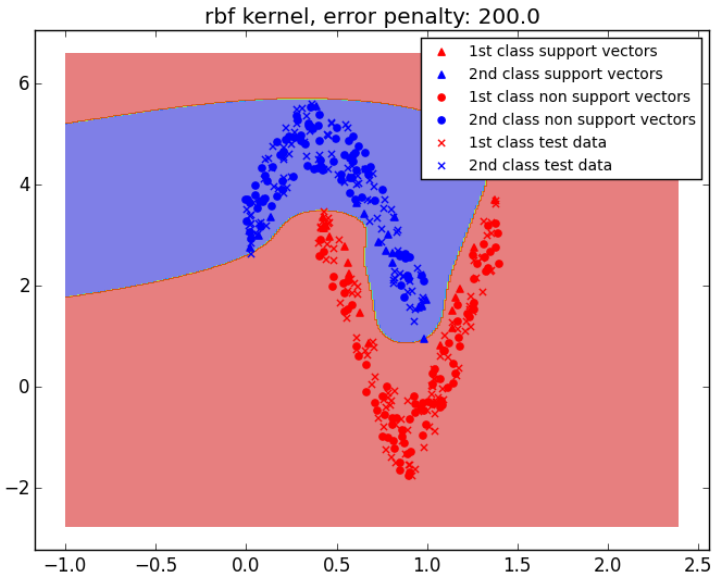

Рис. 4. Оптимальное C. RBF ядро

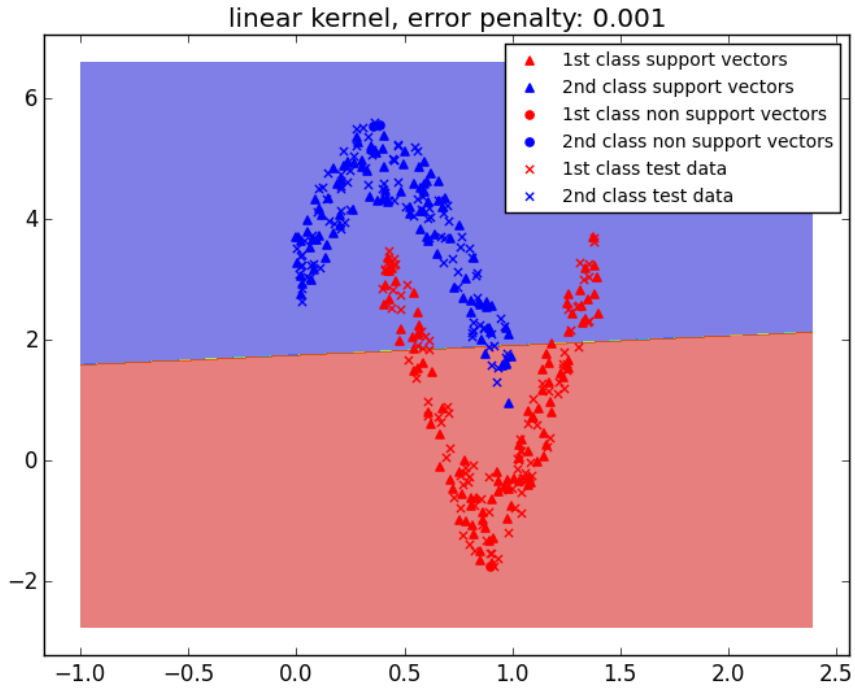

Рис. 5. Неадекватно маленькое C. Линейное ядро

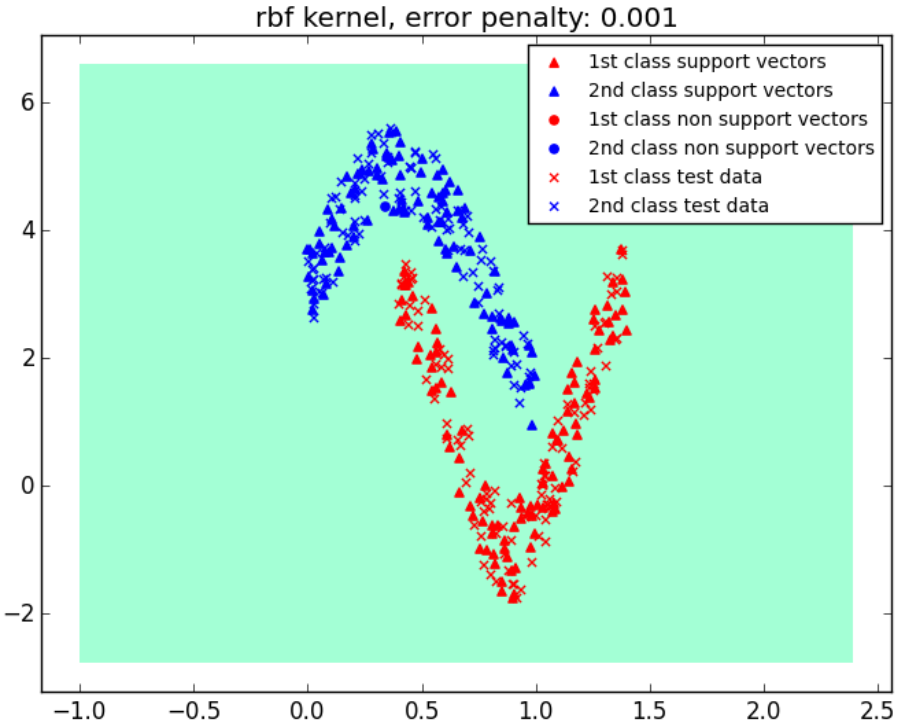

Рис. 6. Неадекватно маленькое C. RBF ядро

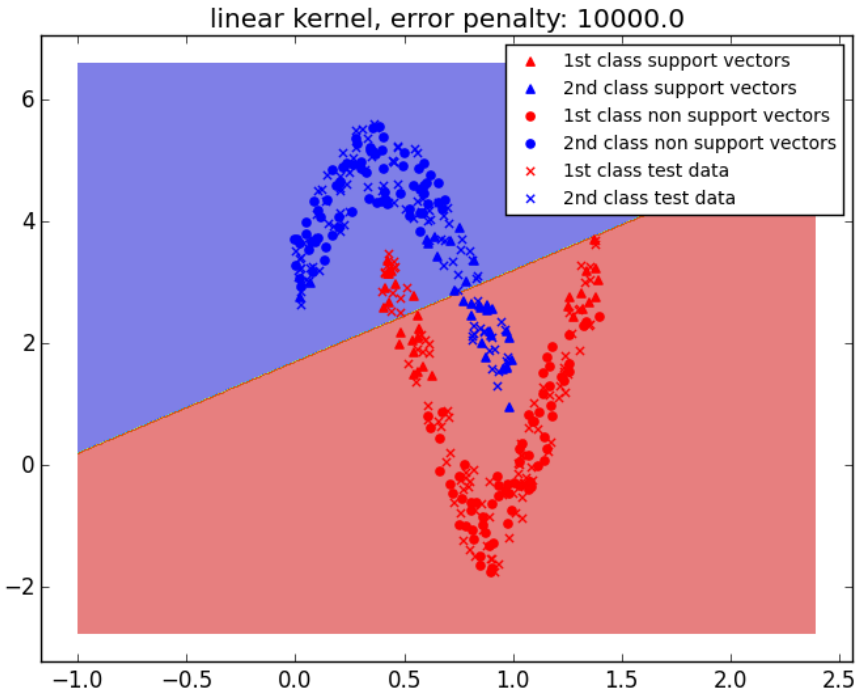

Рис. 7. Неадекватно большое C. Линейное ядро

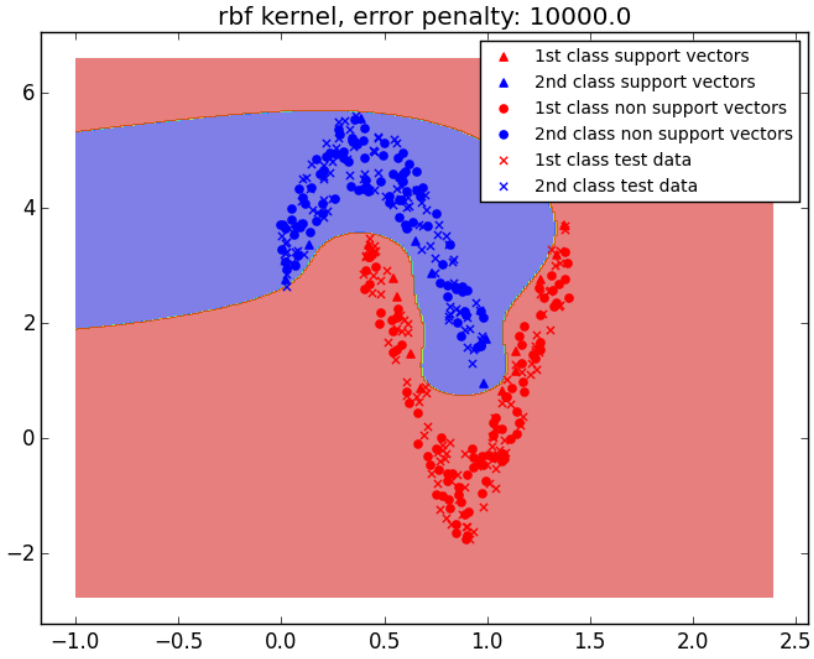

Рис. 8. Неадекватно большое C. RBF ядро

Видно, что в обоих ядрах самая неудачная классификация происходит при условии неадекватно маленького C. В RBF ядре этот случай критичен - все объекты были помечены одним классом.

Также был обучено собственное ядро, определяющее близость между строками (для решения задачи классификации слов). Удалось добиться точности 0.77 на тестовой выборке.

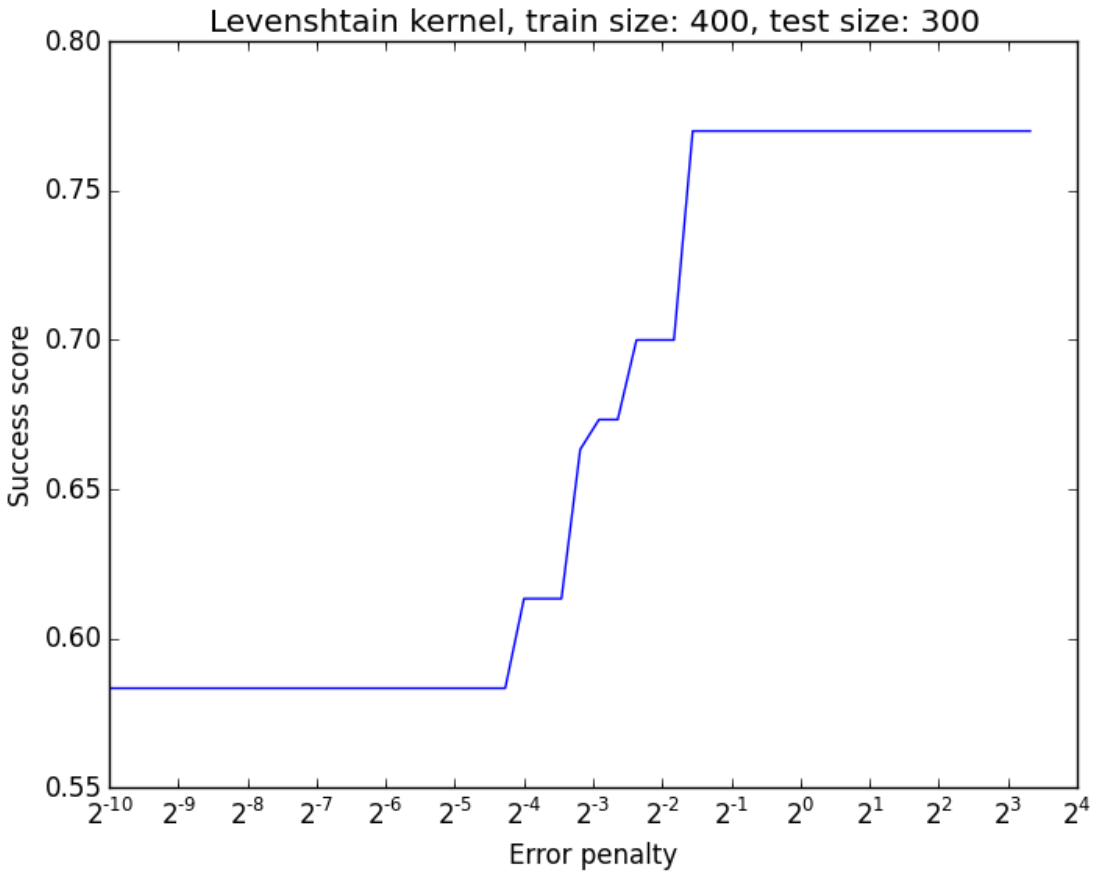

Рис. 8. Точность классификации на тестовой выборке. «Ядро Левенштейна»

СПИСОК ЛИТЕРАТУРЫ

1) **Анализ данных (Программная инженерия)** – http://wiki.cs.hse.ru/Анализ_данных_(Программная_инженерия)

ТЕКСТ ПРОГРАММЫ

```
__author__ = 'Lev Osipov'

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from math import log10
from math import log
from Levenshtein import distance


def learn_stat(train_data, train_labels, test_data, test_labels,
kernel, C):
    clf = SVC(C=C, kernel=kernel)
    clf.fit(train_data, train_labels)
    train_score = clf.score(train_data, train_labels)
    test_score = clf.score(test_data, test_labels)
    support = clf.support_.size
    return train_score, test_score, support


def plot_stat(train_data, train_labels, test_data, test_labels,
kernel, minC, maxC, steps):

    train_score = np.empty(steps)
    test_score = np.empty(steps)
    support = np.empty(steps)

    c_interval = np.logspace(minC, maxC, steps)
    for i, C in enumerate(c_interval):
        train_score[i], test_score[i], support[i] = \
            learn_stat(train_data, train_labels, test_data,
test_labels, kernel, C)
    f, ax = plt.subplots(3, sharex=True)
    i = 0

    ax[i].plot(c_interval, train_score)
    ax[i].set_title("Train score")
    ax[i].set_xscale('log', basex=10)
    i += 1

    ax[i].plot(c_interval, test_score)
    ax[i].set_title("Test score")
    ax[i].set_xscale('log', basex=10)
    i += 1

    ax[i].plot(c_interval, support)
    ax[i].set_title("Number of support vectors")
    ax[i].set_xlabel("Penalty for error")

    ax[i].set_xscale('log', basex=10)
```

```
    plt.show()


def plot_support(train_data, train_labels, test_data,
test_labels, kernel, C):

    clf = SVC(C=C, kernel=kernel)
    clf.fit(train_data, train_labels)

    support_i = clf.support_

    train_sup_vec = train_data[support_i]
    train_sup_lab = train_labels[support_i]

    train_non_sup_vec = np.delete(train_data, support_i, axis=0)
    train_non_sup_lab = np.delete(train_labels, support_i,
axis=0)

    train_sup_vec_1 = train_sup_vec[np.where(train_sup_lab ==
1)]
    train_sup_vec_2 = train_sup_vec[np.where(train_sup_lab !=
1)]

    train_non_sup_vec_1 =
train_non_sup_vec[np.where(train_non_sup_lab == 1)]
    train_non_sup_vec_2 =
train_non_sup_vec[np.where(train_non_sup_lab != 1)]

    test_data_1 = test_data[np.where(test_labels == 1)]
    test_data_2 = test_data[np.where(test_labels != 1)]

    all = np.concatenate((test_data, train_data), axis=0)
    h = .01

    x_min, x_max = all[:, 0].min() - 1, all[:, 0].max() + 1
    y_min, y_max = all[:, 1].min() - 1, all[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    z = z.reshape(xx.shape)
    plt.contourf(xx, yy, z, alpha=0.5)

    plt.scatter(train_sup_vec_1[:, 0], train_sup_vec_1[:, 1],
color='r', marker='^',
                label='1st class support vectors')
    plt.scatter(train_sup_vec_2[:, 0], train_sup_vec_2[:, 1],
color='b', marker='^',
                label='2nd class support vectors')

    plt.scatter(train_non_sup_vec_1[:, 0],
train_non_sup_vec_1[:, 1], color='r', marker='o',
```

```
                label='1st class non support vectors')
    plt.scatter(train_non_sup_vec_2[:, 0],
train_non_sup_vec_2[:, 1], color='b', marker='o',
                label='2nd class non support vectors')

    plt.scatter(test_data_1[:, 0], test_data_1[:, 1], color='r',
marker='x', label='1st class test data')
    plt.scatter(test_data_2[:, 0], test_data_2[:, 1], color='b',
marker='x', label='2nd class test data')

    plt.legend(scatterpoints=1, fontsize=10)
    plt.title("{0} kernel, error penalty: {1}".format(kernel,
C))

    plt.show()
    plt.clf()


def find_optimal_c(train_data, train_labels, test_data,
test_labels, kernel, minC, maxC, steps):
    c_interval = np.logspace(minC, maxC, steps)
    max_score = 0
    c = minC
    for i, C in enumerate(c_interval):
        test_score = learn_stat(train_data, train_labels,
test_data, test_labels, kernel, C)[1]
        if test_score > max_score:
            max_score = test_score
            c = C
    return c

# Task 1
tr_data = pd.read_csv('synth_train.csv').as_matrix()
tr_labels = tr_data[:, 0]
tr_data = np.delete(tr_data, 0, axis=1)

te_data = pd.read_csv('synth_test.csv').as_matrix()
te_labels = te_data[:, 0]
te_data = np.delete(te_data, 0, axis=1)

# CHANGE 'rbf' to 'linear' to see linear kernel results

# Task 2
minimC = log10(1e-2)
maximC = log10(200)
steps_count = 35
plot_stat(tr_data, tr_labels, te_data, te_labels, 'rbf', minimC,
maximC, steps_count)

# Task 3
c = find_optimal_c(tr_data, tr_labels, te_data, te_labels,
'rbf', minimC, maximC, steps_count)
plot_support(tr_data, tr_labels, te_data, te_labels, 'rbf', 1e-
```

```
3)
plot_support(tr_data, tr_labels, te_data, te_labels, 'rbf', c)
plot_support(tr_data, tr_labels, te_data, te_labels, 'rbf', 1e4)

# Levenshtein


def levenshtein_gram_matrix(s1, s2):
    res = np.empty((len(s1), len(s2)))
    for i in range(len(s1)):
        for j in range(len(s2)):
            res[i][j] = 1 / (1 + distance(s1[i], s2[j]))
    return res

with open('en.txt') as f:
    en = f.readlines()

with open('fr.txt') as f:
    fr = f.readlines()

data = np.array(en + fr)

labels = np.concatenate((np.ones(len(en)), -1 *
np.ones(len(fr))))
data = np.column_stack((data, labels))
np.random.shuffle(data)

tr_length = 400
tr_data = data[:tr_length]
tr_labels = tr_data[:, 1]
tr_data = tr_data[:, 0]

te_length = 300
te_data = data[tr_length:tr_length+te_length]
te_labels = te_data[:, 1]
te_data = te_data[:, 0]

train_gram_matrix = levenshtein_gram_matrix(tr_data, tr_data)
test_gram_matrix = levenshtein_gram_matrix(te_data, tr_data)

c_range = np.logspace(log(0.001, 2), log(10, 2), base=2)
scores = []
for C in c_range:
    clf = SVC(C=C, kernel='precomputed')
    clf.fit(train_gram_matrix, tr_labels)
    scores.append(clf.score(test_gram_matrix, te_labels))

print("Best: {0}".format(max(scores)))

plt.plot(c_range, scores)
plt.xscale('log', basex=2)
plt.xlabel("Error penalty")
plt.ylabel("Success score")
```

```
plt.title("Levenshtain kernel, train size: {0}, test size:
{1}".format(tr_length, te_length))
plt.show()
```