Klasifikacija besedil z jedrnimi metodami

Tomaž Tomažič (63100281)

14. junij 2015

1 Uvod

V nalogi sem uporabili SVM za klasifikacijo besedil da bi pri tem spoznal pomen jeder.

2 Podatki

Podatke smo si morali sami poiskati v treh kategorijah. Shranil sem bloge. Izbral sem si dve podobni kategoriji Windows blog (A) in Bing blog (B). Za drugacno temo pa sem izbral Zoella blog (C), ki je popularen blog za zenske, licenje in podobne reci.

3 Metode

Potrebno je bilo napisati jedro za podporne vektorje implementirane v sklearn knjižnjici. Jedro deluje na principu merjenja razdalje s kompresijo, kot je bilo podano v navodilih naloge. Razdalje sem obrnil tako da sem odstel vse vrednosti od najvecje v matriki razdalj. Napovedi sem naredil za vse tri kombinacije med kategorijami: A in B, A in C, B in C. Napoved sem naredil tako, da sem se naucil model na 80% podatkih, ostalih 20% pa sem uporabil za napoved verjetnosti razreda.

4 Rezultati

Rezultate sem preveril kar z metodo ostrega očesa, ko sem izpisal napovedane vrjetnosti. Rezultati so bili nekoliko boljsi ko sem napovedoval cisto razlicne razrede (A-C in B-C) v primerjavi z podobnimi razredi (A-B). Menim da bi bilo to mozno se precej izboljsati z iskanjem pravih parametrov.

5 Izjava o izdelavi domače naloge

Domačo nalogo in pripadajoče programe sem izdelal sam.

Priloge

A Programska koda

```
import zlib
import copy
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score, accuracy_score, auc

class SVM:
    def __init__(self, data):
        self.learner = SVC(C = 1.1, kernel=self.string_kernel, probability=True
    )
        self.data = data

    def Z(self, a):
        return len(zlib.compress(a.encode()))

    def distance(self, a, b):
        Z = self.Z
        a = self.data[int(a)]
        b = self.data[int(b)]

        ab = a+b
        ba = b+a
        return (Z(ab) - Z(a)) / Z(a) + (Z(ba) - Z(b)) / Z(b)

    def string_kernel(self, X, Y):
        M = np.zeros((len(X), len(Y)))
        for i, a in enumerate(X):
            for j, b in enumerate(Y):
                M[i, j] = self.distance(a[0], b[0])
        return M.max() - M

    def fit(self, X, Y):
        return self.learner.fit(X, Y)

    def predict(self, X):
        return self.learner.predict_proba(X)

data_a = [ open("source/windows/"+str(i)+".txt", "rt").read() for i in range(1,
     21)]
data_b = [ open("source/bing/"+str(i)+".txt", "rt").read() for i in range(1,
    21)]
data_c = [ open("source/zoella/"+str(i)+".txt", "rt").read() for i in range(1,
    21)]

def test(data_a, data_b):
    da = data_a[:-4]
    db = data_b[:-4]
    data_ab = copy.deepcopy(da)
    data_ab.extend(db)

    Xab = np.vstack(np.r_[0:len(data_ab)])
    Yab = np.array([0]*len(da) + [1]*len(db))

    s = SVM(data_ab)
```

```
    s.fit(Xab, Yab)
    s.data.extend(data_a[-4:])
    s.data.extend(data_b[-4:])

    return s.predict(np.vstack(np.r_[len(da)*2:len(data_a)*2]))

print(test(data_a, data_b))
print(test(data_a, data_c))
print(test(data_b, data_c))
```