Bootstrapping the Support Vector Machine

Thomas Goerttler[1,2,3], Christian Koopmann[1,2,3], Patricia Craja[1,2,3]

[1]Humboldt University of Berlin, Unter den Linden 6, 10099 Berlin
[2]Free University Berlin, Kaiserswerther Str. 16-18, 14195 Berlin
[3]Technische Universitt Berlin, 10623 Berlin
thomas.goerttler@gmail.com
c.k.e.koopmann@gmail.com
Patricia.craja@gmx.de

Abstract

Support Vector Machines are one of the most successful methods of Machine Learning. After training they can predict for each point a possible class. Unfortunetly there is not uncertaincy captured, that the point is really in this class. We found out a way to calculate variance for a support vector machine. We use bootstrapped samples and take this into account. The variance can be calculated parallized.

Introduction

Support Vector Machines are one of the most successful methods of Machine Learning. By reducing non-linear complex decisions problems to linear problems through application of the kernel-Trick, they represent a computationally efficient way to tackle these problems.

svm uses hyperlance space feature to seperate data. The dimension of feature space ist controlled by choice of kernel function. Commen Kernels are the linear kernel, the gausian kernel (rbf) and the polynomial kernel.

SVMs based on certain kernels (e.g. Gaussian RBF Kernel) are non parametric methods. Since the distribution of the underlying data is generally unknown so is the finite sample distribution of these methods. There has been considerable research on the asymptotic distribution of SVMs, which have been shown to be asymptotically normally distributed under certain con- ditions. An alternative idea to estimate these distributions is using Efrons empirical bootstrap. The idea behind this method is to repeatedly draw samples with replacement from the full data according to the empirical distribution function of the data. Through the repeated calculation of the statistic of interest one can get an estimate of its distribution. For the SVM this estimate has been shown to be consistent under relatively mild conditions. ..

Background

What is the status in the science world right now. Important has been **bootstrap, distances and parallelization**

Problem

There is noy way to calculate uncertaincy (variance) of the prediction as we have no assuptions about the distribution.We found out that we can sample variance by bootsrapping. Futhermore we found out that they are tuning parameter, data-distribution option and svm attributes that influce this variance.

Predictions are only binary variables and therefore might be identical across all bootstrap samples Therefore we use minimal distance of predictionpoint to decision boundary as real valued substitue

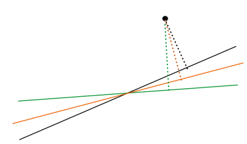
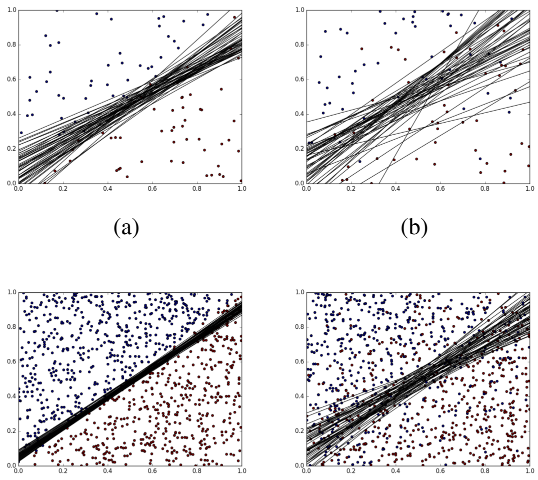


Figure 1: Really awesome trick of distance

Implementation

Kernels are powerful! Ker-



(a) (b) (c) (d)

Figure 2: (a) They are 100 perfectly seperated points, (b) They are 100 not perfectly seperated points ......

nels are powerful! Kernels are pow-

```
def bootstrap_the_svm(trainings_data,
    prediction_data, kernel, C, gamma = "
    auto", degree = 3, processes = 10,
    replications = 1000):

    input_parameters = SVM_Input(
        trainings_data, prediction_data,
        kernel, C, gamma, degree)
    data = input_parameters
    real_svm = do_svm(data)

    ### Do bootstrapping
    PROCESSES = processes
    REPLICATIONS = replications
    pool = Pool(processes = PROCESSES)
    results = pool.map(single_sample_and_svm
        , [data] * REPLICATIONS)

    ### Calculate the Variance of the
        Support Vector Machine

    points_information = Points_Information(
        results)

    variance_of_svm_probabilites =
        calculate_variance_of_svm(
        points_information.probabilites)
    variance_of_svm_distance_to_hyperplane =
         calculate_variance_of_svm(
        points_information.distances)

    result = Bootstrap_Result([real_svm[0],
        real_svm[1], real_svm[2],
        variance_of_svm_probabilites,
        variance_of_svm_distance_to_hyperplane
        ], real_svm[0].n_support_ )

    return(result)
```

Figure 3: Implementation of Bootstrapping

erful! Kernels are powerful!

Results

There we three indicators we found out. The C Parameter, Balance and number of support vector. Once we used linear, once we used gaussian.

Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels are powerful! Kernels

```
def get_data_normaly_distributed(coefs, errorCoef, intercept, size):
  inputs = []
  error = errorCoef*rd.standard_normal(size)
  y = error + intercept
  for i in range(len(coefs)):
     inputs = inputs + [rd.standard_normal(size)]
     y = y+coefs[i]*inputs[i]
  y = sign(y)
  inputs = list(zip(*inputs))
  return([y,inputs])
```

Figure 4: Implementation of Datasimulation normally distributetd

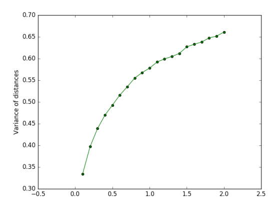
```
def get_data_centroid_distributed(coefs, locations, errorCoef, size, intercept, distance, xdistribution = "normal", par1 = 0, par2 = 1):
  X = []
  dimension = len(list(zip(*locations)))
  for i in range(dimension):
    if(xdistribution == "normal"):
      X = X + [rd.normal(par1, par2, size)]
    elif(xdistribution == "uniform"):
      X = X + [rd.uniform(par1, par2, size)]
    else:
      print("Please choose supported Distribution")
      return None
  X = list(zip(*X))
  distances = []
  error = errorCoef*rd.standard_normal(size)
  y = error + intercept
  for i in range(size):
    newDistance = pdist([X[i]]+locations, distance)
    newDistance = newDistance[:len(locations)]
    inverseDistance = power(newDistance, -1)
    y[i] = y[i] + dot(coefs, inverseDistance)
    distances = distances + [newDistance]
  y = sign(y)
  #distances = list(zip(*distances))
  return([y,X])
```

Figure 5: Implementation of Datasimulation normally distributetd

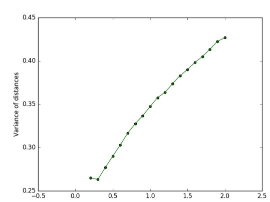
are powerful! Kernels are powerful!

Influence of the tuning parameter C

Kernels are powerful! Kernels l?! Kernels are powerful?! Kernels are powerful?! Kernels are powerful?! Kernels are powerful?! Kernels are powerful?! Kernels are powerful?! Kernels are powerful?!ls are powerful?! Kernels are powerful?! Kernels are powerf



(a)

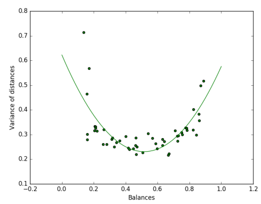


(b)

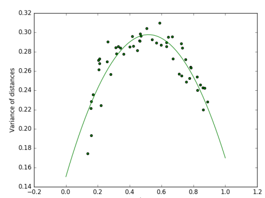
Figure 6: The results for chaning Balances linear Datasize 100 Repl 1000 Datasets 20

Influence of the balance of the training dataset

Kernels are powerful! Kernels are



(a)



(b)

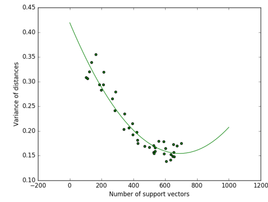
Figure 7: The results for chaning Balances linear Datasize 500 Repl 1000

Influence of the svm attributes

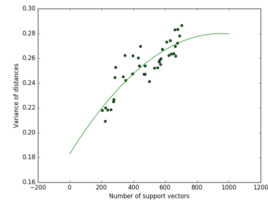
Kernels are powerful! Kernels are powerful!

Conclusion

Nice to work with. I also works whit real datasets



(a)



(b)

Figure 8: The results for chanching number of support vectors Datasize 1000 Repl 500

Influence of different aspects on variance

C-Parameter: Positive but decreasing influence on variance. Balance of Data: Positive Quadratic influence for Linear and negative quadratic influence for Gaussian SVM. Both with extremum around 0.5 (perfect balance). Number of support vectors: Positive influence for Gaussian and negative influence for linear SVMs

Outlook

Analyse influence of dimensionality on variances. Analyse Data simulated from more ”exotic” distributions. Analyse influence of other tuning parameters e.g. ”Gamma” of rbf-kernel, degree of polynomial kernel etc.

References