Final Course Assignment

Mark Niehues, Stefaan Hessmann
Mathematical Aspects in Machine Learning

13. Juli 2017

1 Introduction

In the past course we dealt with the broad mathematical foundations of machine learning. To get an idea of what the consequences of those mathematical theorems and approaches are and to get in touch with the standard Python tools, we have evaluated an comparatively easy data science example found on `kaggle.com`. Since this was our first machine learning project, we decided to deal with an rather simple problem. The example dataset [1] consists of the historic passenger records of the disastrous Titanic maiden voyage in 1912. The goal in this Challenge was to predict if a passenger survived the accident based on informations like for example age, sex and payed ticket fare. Therefore it is in terms of machine learning a **classification problem** with two classes: survived and not survived.

Inspired by sample solutions from the website, we first took a deeper look on the dataset and tried to select the most significant influences by reviewing the statistical properties of the dataset. In the following we implemented an naive Sequential Minimal Optimization (SMO) algorithm and ran a few tests with them in order to finally compare the results with other machine learning algorithms.

2 Applying Machine Learning Methods on the Titanic Disaster

blalba

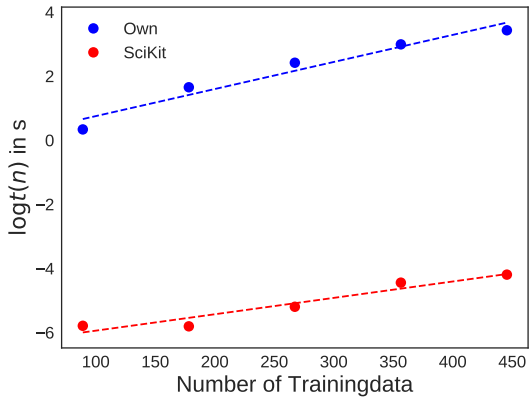

Abbildung 1: Benchmark

3 Implementation of an easy SMO Algorithm

To get a better understanding of what a Support Vector Machine does, we decided to implement one on our own using several publications. Most of them were based on the important paper of Platt [2] where he introduced a new approach for the calculation of the Support Vectors that improves the performance a lot. This algorithm is called Sequential Minimal Optimization.

4 Appendix

Listing 1: Hello World

```
#   Copyright (C) 2017 Mark Niehues, Stefaan Hessmann
#
#   This program is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#


import numpy as np


class Kernels:
    """
    Class that holds different Kernels
    """
    def __init__(self, gamma):
        self.gamma = gamma
        self.kernels = {
            "rbf" : self.kernel_rbf,
            "linear": self.kernel_lin}

    def get_kernel(self, kernel_name):
        return self.kernels[kernel_name]

    def kernel_lin(self, x, y):
        """
        Linear kernel
        """
        return x.dot(y)

    def kernel_rbf(self, x, y):
        """
        RBF Kernel
        """
        d = x - y
        return np.exp(-np.dot(d, d) * self.gamma)
```

Input:

C: regularization parameter
tol: numerical tolerance
max_passes: max # of times to iterate over α's without changing
$(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$: training data

Output:

$\alpha \in \mathbb{R}^m$: Lagrange multipliers for solution
$b \in \mathbb{R}$: threshold for solution

- Initialize $\alpha_i = 0, \forall i, \quad b = 0$.
- Initialize $passes = 0$.
- **while** $(passes < max\_passes)$
  - $num\_changed\_alphas = 0$.
  - **for** $i = 1, \ldots m$,
    - Calculate $E_i = f(x^{(i)}) - y^{(i)}$ using (2).
    - **if** $((y^{(i)}E_i < -tol \quad \&\& \quad \alpha_i < C) \;||\; (y^{(i)}E_i > tol \quad \&\& \quad \alpha_i > 0))$
      - Select $j \neq i$ randomly.
      - Calculate $E_j = f(x^{(j)}) - y^{(j)}$ using (2).
      - Save old α's: $\alpha_i^{(\text{old})} = \alpha_i$, $\alpha_j^{(\text{old})} = \alpha_j$.
      - Compute L and H by (10) or (11).
      - **if** $(L == H)$
        continue to next i.
      - Compute η by (14).
      - **if** $(\eta >= 0)$
        continue to next i.
      - Compute and clip new value for α_j using (12) and (15).
      - **if** $(|\alpha_j - \alpha_j^{(\text{old})}| < 10^{-5})$
        continue to next i.
      - Determine value for α_i using (16).
      - Compute b_1 and b_2 using (17) and (18) respectively.
      - Compute b by (19).
      - $num\_changed\_alphas := num\_changed\_alphas + 1$.
    - **end if**
  - **end for**
  - **if** $(num\_changed\_alphas == 0)$
    $passes := passes + 1$
  - **else**
    $passes := 0$
- **end while**

Abbildung 2: Pseudo Code of the SMO algorithm. Taken from: [3]

Literatur

[1] Kaggle. *Titanic: Machine Learning from Disaster*. 13. Juli 2017. URL: https://www.kaggle.com/c/titanic.

[2] John Platt. „Sequential minimal optimization: A fast algorithm for training support vector machines“. In: (1998).

[3] Unknown. *The Simplified SMO Algorithm*. 13. Juli 2017. URL: http://cs229.stanford.edu/materials/smo.pdf.