Ostfalia
Hochschule für angewandte
Wissenschaften

Fakultät Informatik

Infrastruktur-agnostische Entwicklung und Bereitstellung von Webanwendung

Niklas Röske
Matrikel-Nr. 70456600

Masterarbeit im Studiengang Informatik
zur Erlangung des akademischen Grades:
Master of Science

Ostfalia Hochschule für angewandte Wissenschaften

1. Prüfer: Prof. Dr. Hans Grönniger
2. Prüfer: Prof. Dr. Bernd Müller

Salzgitter · Suderburg · **Wolfenbüttel** · Wolfsburg

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

Ort, Datum

Unterschrift

Abstract

Your abstract goes here..

Inhaltsverzeichnis

Abkürzungsverzeichnis

- OSC...Orthographic Star Coordinates
- SC...Star Coordinates
- CO...Composition Operators
- LSS...Least Square Solution
- DSC...Distance Consistency
- CD...Centroid Density
- CDC...Centroid Distance Change
- VML...Visual Machine Learning

1. Einleitung

1.1. Hintergrund und Motivation

1.2. Zielsetzung der Arbeit

1.3. Forschungsfragen

2. Theoretischer Hintergrund

2.1. Web-Anwendungen

2.2. Infrastruktur unabhängige Entwicklung

2.3. Herausforderungen bei der Entwicklung

[2]
[1]

3. Methodik

3.1. Auswahl Entwicklungstechnologien

3.2. Architektur und Design der Web-Anwendung

3.3. Implementierung un Testing?

4. Empfehlungen und bewährte Praktiken

4.1. Best Practices

4.2. Technische Aspekte

5. Fallstudie

5.1. Beschreibung der Anwendung

5.2. Bewertung der Ergebnisse

6. Diskussion

6.1. Wichtigste Erkenntnisse

6.2. Beantwortung der Forschungsfrage

6.3. Kritische Bewertung der EMpfehlung

7. Ausblick

Your discussion goes here ...

Literaturverzeichnis

[1] Demartines, P. ; Herault, J.: Curvilinear component analysis: a self-organizing neural network for nonlinear mapping of data sets. In: *IEEE transactions on neural networks* 8 (1997), Nr. 1, S. 148–154. – ISSN 1045-9227

[2] Sammon, J. W.: A nonlinear mapping for data structure analysis. In: *IEEE Transactions on Computers* 18 (1969), Nr. 5, S. 401–409

Anhang

Anhang A:

Pseudocode for Heuristics

Heuristic 1: Random Selection Shift RSS

```
function heuristic_random_selection_shift(df_p, iterator, num_classes):
    Data: df_p is the dataframe in projection space, iterator is the current iteration
          step, num_classes is the amount of different classes in the dataset
    Result: dp shift vector, selected_class to be shifted, calculated DSC_value,
            calculated CD_value, calculated total_dist
    /* extract star coords and class of data to a new dataframe                */
    df_star = df_p[['X','Y','class']]
    /* calculate class centroids and save them in a new dataframe              */
    df_centroids = df_star.groupby('class', sort=True).mean().reset_index()
    /* calculate coordinates of the central centroid                           */
    central_centroid = df_centroids[['X', 'Y']].mean()
    /* call a function to calculate all distances between centroids            */
    df_distances = calc_centroid_distances(df_centroids)
    /* calculate the distance from each point to its associated centroid       */
    df_centroid_distances = calc_dist_p_to_assoc_centroid(df_centroids, df_star)
    /* calculate CD, DSC and total_dist for result                             */
    CD_value = df_centroid_distances['distance'].sum()
    DSC_value = calc_dsc(df_centroids, df_star)
    total_dist = df_distances['distance'].sum()
    /* randomly choose a class to be shifted                                   */
    selected_class = np.random.randint(num_classes)
    /* select all distances for selected class                                 */
    selected_class_distances = df_distances.(df_distances[selected_class])
    /* other_class is the nearest class to selected_class                      */
    min_selected_class_distance = selected_class_distances['distance'].idxmin()
    /* calculate the new shifting vector dp                                    */
    dp = calc_dp(df_centroids, selected_class, other_class, central_centroid, num_iter)
```

Abbildung 1:: Pseudocode for RSS

Heuristic 2: Order Selection Shift - OSS

```
function heuristic_order_selection_shift(df_p, iterator, num_classes):
    Data: df_p is the dataframe in projection space, iterator is the current iteration
          step, num_classes is the amount of different classes in the dataset
    Result: dp shift vector, selected_class to be shifted, calculated DSC_value,
            calculated CD_value, calculated total_dist
    /* extract star coords and class of data to a new dataframe                  */
    df_star = df_p[['X','Y','class']]
    /* calculate class centroids and save them in a new dataframe                */
    df_centroids = df_star.groupby('class', sort=True).mean().reset_index()
    /* calculate coordinates of the central centroid                             */
    central_centroid = df_centroids[['X', 'Y']].mean()
    /* call a function to calculate all distances between centroids              */
    df_distances = calc_centroid_distances(df_centroids)
    /* calculate the distance from each point to its associated centroid         */
    df_centroid_distances = calc_dist_p_to_assoc_centroid(df_centroids, df_star)
    /* calculate CD, DSC and total_dist for result                               */
    CD_value = df_centroid_distances['distance'].sum()
    DSC_value = calc_dsc(df_centroids, df_star)
    total_dist = df_distances['distance'].sum()
    /* choose a class to be shifted by order                                     */
    selected_class = num_iter % num_classes)
    /* select all distances for selected class                                   */
    selected_class_distances = df_distances.(df_distances[selected_class])
    /* other_class is the nearest class to selected_class                        */
    min_selected_class_distance = selected_class_distances['distance'].idxmin()
    /* calculate the new shifting vector dp                                      */
    dp = calc_dp(df_centroids, selected_class, other_class, central_centroid, num_iter)
```

Abbildung 2:: Pseudocode for OSS

Heuristic 3: Point Selection Shift - PSS

```
function heuristic_order_selection_shift(df_p, iterator, num_classes):
    Data: df_p is the dataframe in projection space, iterator is the current iteration
          step, num_classes is the amount of different classes in the dataset
    Result: dp shift vector, selected_class to be shifted, calculated DSC_value,
            calculated CD_value, calculated total_dist
    /* extract star coords and class of data to a new dataframe                 */
    df_star = df_p[['X','Y','class']]
    /* calculate class centroids and save them in a new dataframe               */
    df_centroids = df_star.groupby('class', sort=True).mean().reset_index()
    /* calculate coordinates of the central centroid                            */
    central_centroid = df_centroids[['X', 'Y']].mean()
    /* call a function to calculate all distances between centroids             */
    df_distances = calc_centroid_distances(df_centroids)
    /* calculate the distance from each point to its associated centroid        */
    df_centroid_distances = calc_dist_p_to_assoc_centroid(df_centroids, df_star)
    /* calculate CD, DSC and total_dist for result                              */
    CD_value = df_centroid_distances['distance'].sum()
    DSC_value = calc_dsc(df_centroids, df_star)
    total_dist = df_distances['distance'].sum()
    /* find the maxmimum distance                                               */
    max_dist_idx = df_centroid_distances['distance'].idxmax()
    /* select the point that is to be shifted                                   */
    centroid_id = df_centroid_distances.loc[max_dist_idx, 'class']
    centroid_coords = [df_centroids.loc[centroid_id, 'X'], df_centroids.loc[centroid_id,
     'Y']]
    point_coord = [df_centroid_distances.loc[max_dist_idx, 'X'],
     df_centroid_distances.loc[max_dist_idx, 'Y']]
    /* create noise                                                             */
    noise = np.random.normal(0, 1, 1) * 100 / (1000 + num_iter)
    /* calculate the new shifting vector dp                                     */
    dp = [centroid_coords[0] - point_coord[0] + noise, centroid_coords[1] -
     point_coord[1] + noise]
```

Abbildung 3:: Pseudocode for PSS

Anhang B:

Key values for all heuristics for each dataset

dataset	DSC_{start}	DSC_{end}	CD_{start}	CD_{end}	$d_{c,total,start}$	$d_{c,total,end}$
ecoli	63.88%	65.07%	11636	11651	3073	3086
iris	89.93%	89.93%	3585	3498	340	358
statlog	21.06%	28.76%	65743	65963	112	222
wdbc	86.27%	94.72%	22820	16437	79	66
wine	72.32%	92.66%	7703	7539	201	353
yeast	27.44%	27.38%	44547	44911	2613	2635

Tabelle 1:: Key values for RSS for each dataset

dataset	DSC_{start}	DSC_{end}	CD_{start}	CD_{end}	$d_{c,total,start}$	$d_{c,total,end}$
ecoli	63.88%	63.88%	11636	11652	3073	3078
iris	89.93%	90.6%	3585	3483	340	364
statlog	21.06%	22.76%	65743	65784	112	132
wdbc	86.27%	94.72%	22820	16399	79	65
wine	72.32%	93.22%	7703	7549	201	357
yeast	27.44%	27.44%	44547	44713	2613	2616

Tabelle 2:: Key values for OSS for each dataset

dataset	DSC_{start}	DSC_{end}	CD_{start}	CD_{end}	$d_{c,total,start}$	$d_{c,total,end}$
ecoli	63.88%	67.76%	11636	11545	3073	3136
iris	89.93%	93.29%	3585	3111	340	444
statlog	21.06%	41.27%	65743	62631	112	415
wdbc	86.27%	96.3%	22820	18776	79	98
wine	72.32%	93.22%	7703	7389	201	384
yeast	27.44%	28.32%	44547	46156	2613	2765

Tabelle 3:: Key values for MSS for each dataset

dataset	DSC_{start}	DSC_{end}	CD_{start}	CD_{end}
ecoli	63.88%	63.39%	11636	11729
iris	89.93%	89.93%	3585	3585
statlog	21.06%	22.51%	65743	57698
wdbc	86.27%	91.37%	22820	17566
wine	72.32%	100%	7703	5641
yeast	27.44%	24.81%	44547	43926

Tabelle 4:: Key values for PSS for each dataset