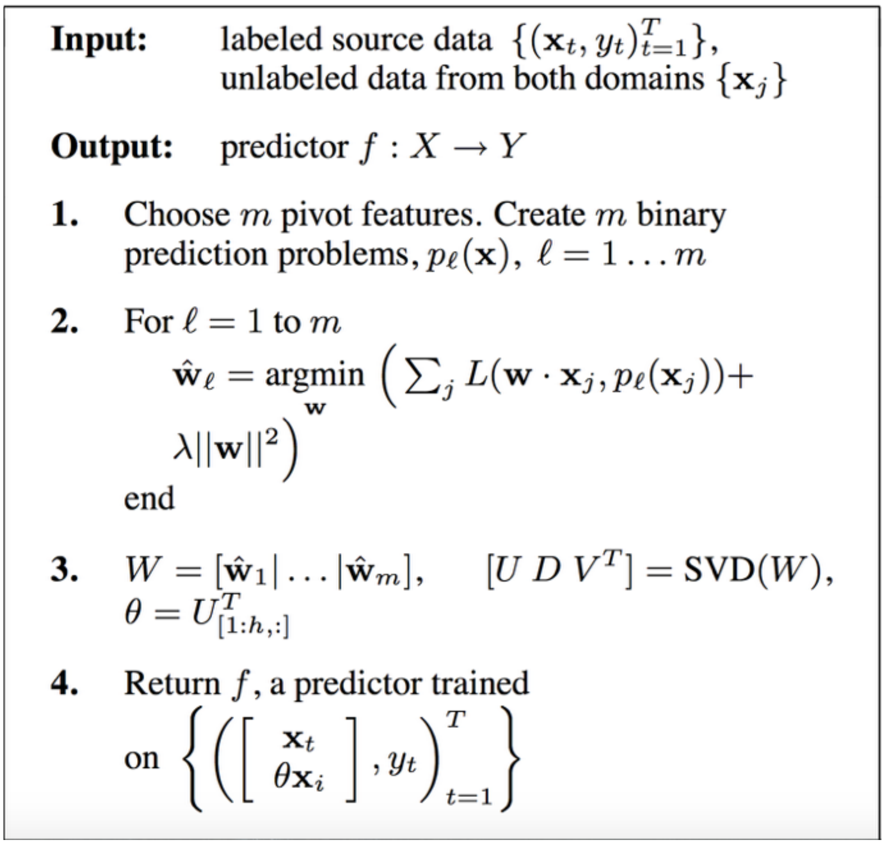
SCL Algorithm for Domain Adaptation proposed by Blitzer et al (2006).

SCL Algorithm Overview from Blitzer et al. (2006):



```
Input:   labeled source data $\{(\mathbf{x}_t, y_t)_{t=1}^{T}\}$,
         unlabeled data from both domains $\{\mathbf{x}_j\}$
Output:  predictor $f : X \to Y$
1.  Choose $m$ pivot features. Create $m$ binary
    prediction problems, $p_\ell(\mathbf{x}),\ \ell = 1 \dots m$
2.  For $\ell = 1$ to $m$
        $\hat{\mathbf{w}}_\ell = \underset{\mathbf{w}}{\operatorname{argmin}} \left( \sum_j L(\mathbf{w} \cdot \mathbf{x}_j, p_\ell(\mathbf{x}_j)) + \lambda ||\mathbf{w}||^2 \right)$
    end
3.  $W = [\hat{\mathbf{w}}_1 | \dots | \hat{\mathbf{w}}_m]$,    $[U\ D\ V^T] = \mathrm{SVD}(W)$,
    $\theta = U^T_{[1:h,:]}$
4.  Return $f$, a predictor trained
    on $\left\{ \left( \begin{bmatrix} \mathbf{x}_t \\ \theta \mathbf{x}_i \end{bmatrix}, y_t \right)_{t=1}^{T} \right\}$
```

My Explanation of the algorithm:

What we have: a little amount of the labelled data from one domain (so-called source domain), e.g. reviews of computers and big amount of the unlabelled data from other domain (so-called target domain), e.g. reviews of cell phones.

What we want: to adapt data/features(SCL is feature-based algorithm) from source domain into target domain, so that we can train the model on the little amount of labelled source data and apply this model to the unlabelled target data. Performance of the model trained on one domain drops if we apply it directly to the data from the other domain, due to adaptation error.

The main reason, why we have this adaptation error, is: that many features in the source are not longer useful for the target. An example from Blitzer et al. (2006) for sentiment analysis, the feature „*fast dual-core*" from source domain is completely different from „*good-quality reception*" from target domain (we will call such distinct features non-pivots). But, good news with SCL we can capture such differences.

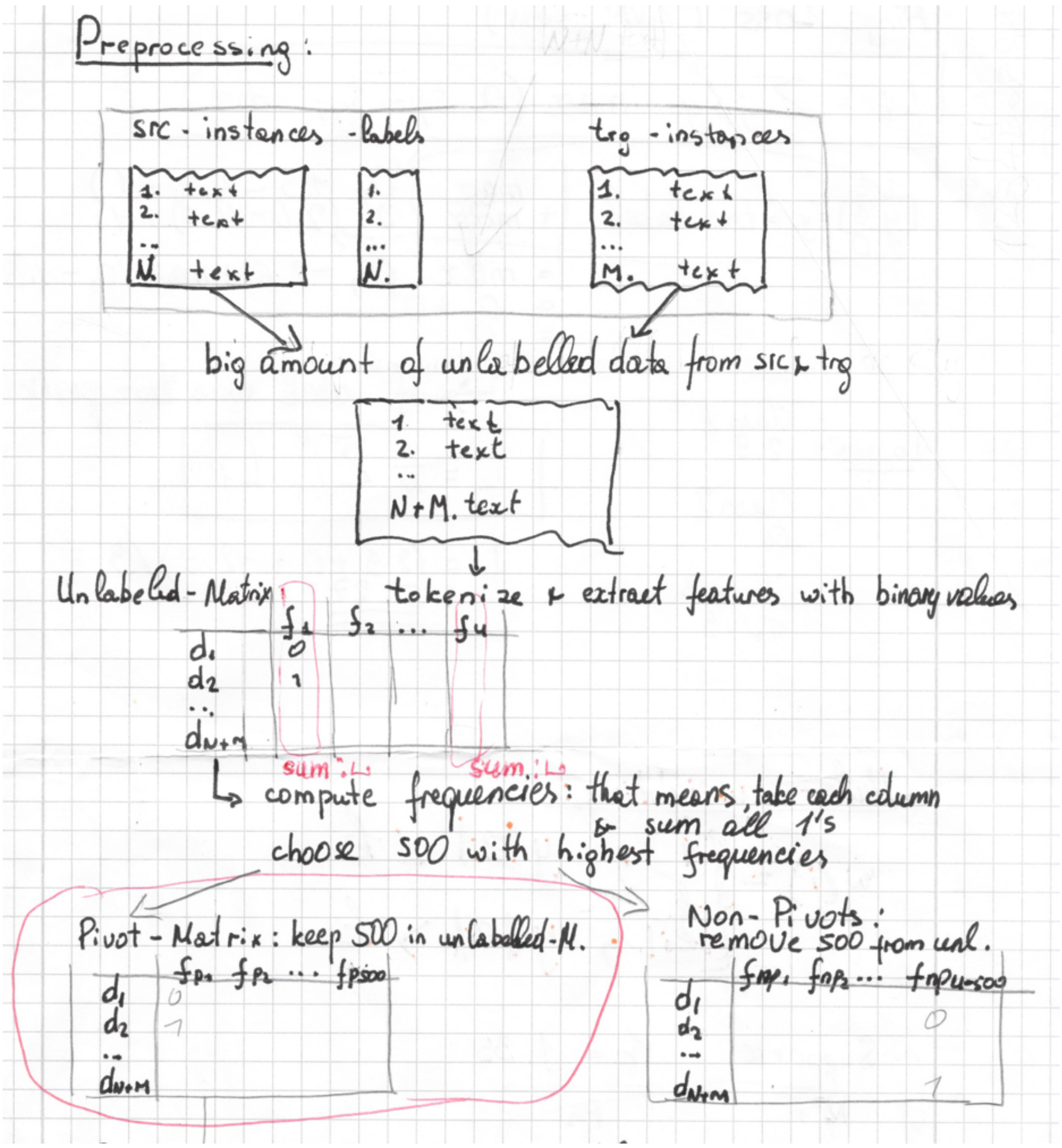
The main idea of SCL is: to find out such features from both (source and target) domains, which behave „similar", e.g.: the features „*excellent*", „*awful*" behave in both domain similar, i.e. „*excellent*" is positive and „*awful"* is negative. We will call the features with similar behavior pivot-features. So if the completely distinct features like „*fast dual-core*" from computer domain and „*good-quality reception*" from cell phones domain have hight correlation with pivot-feature „*excellent*" and low correlation with „*awful*" they can be captured in the same way during the training. We will call the correlation detection between pivot and non-pivot the pivot-prediction. And the good thing there is, that we don't need any labelled data from target domain (only maybe for evaluation :-).

So we have already three terms: Pivots, Non-Pivots and Pivot-Prediction. If we have all this three parts, we can train our model and test it on the target data. But first, we will talk about how to get these three parts.

1. Step: Choose *m* pivot features and create *m* binary prediction problems:

- Put all data from source and target together and extract the features, e.g. unigrams and bigrams;
- determine frequencies or compute Mutual Information (between features and source labels) of the features and sort them in descending order;
- choose 500 first features with highest values, e.g. with highest frequencies. Congratulation, you have your pivots. Namely document-term-matrix, terms are pivots and documents are all instances from unlabelled data, and the weights of the matrix are 1 or 0 (=binary);
- the remaining features are non-pivots.

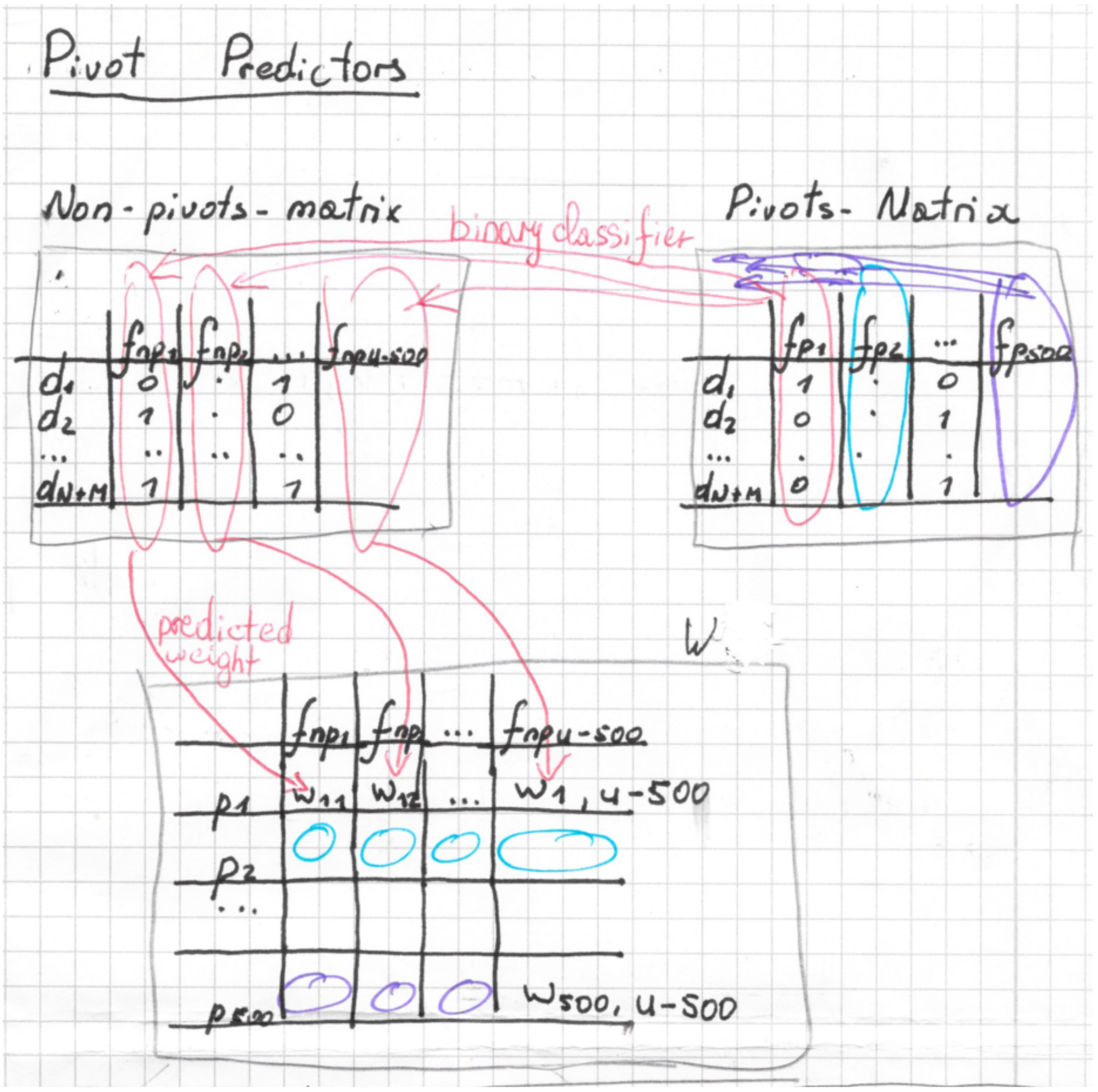
- Visualization of the extraction/implementation of the pivots and non-pivots:



- With this both pivots- and non-pivots-matrixes we can create now *m* binary prediction problems.

- What is this binary prediction problem? So as I wrote above, we will correlate pivots with non-pivots, e.g. „*excellent*“ with „*fast dual-core*“ and „*good-quality reception*“. That means, our binary classifier, e.g. SGD (Stochastic Gradient Descent)) will decide whether this pivot feature appear with this non-pivot and with next non-pivot … . If many different non-pivots from both domains are hight correlated (=have hight predicted weights) with the same pivot, then these non-pivots are assumed to be the same. Binary classification problem is also in the form: "*Does this particular pivot feature occur in this instance?*" (Blitzer et al., 2006), where values(0 or 1) are served as labels of predictions.

- For the binary classification we need something where we can store the predicted weights. Therefor we will create an empty matrix and fill it than with weights for each binary prediction, we will call it „*W*“. The axis 0 is a number of pivots and axis 1 is the names of non-pivots.
- We fill now our *W* with predicted weights.
- Visualization of the pivot predictors:



2. Step: For *l = 1 to m*, detect *each w:*

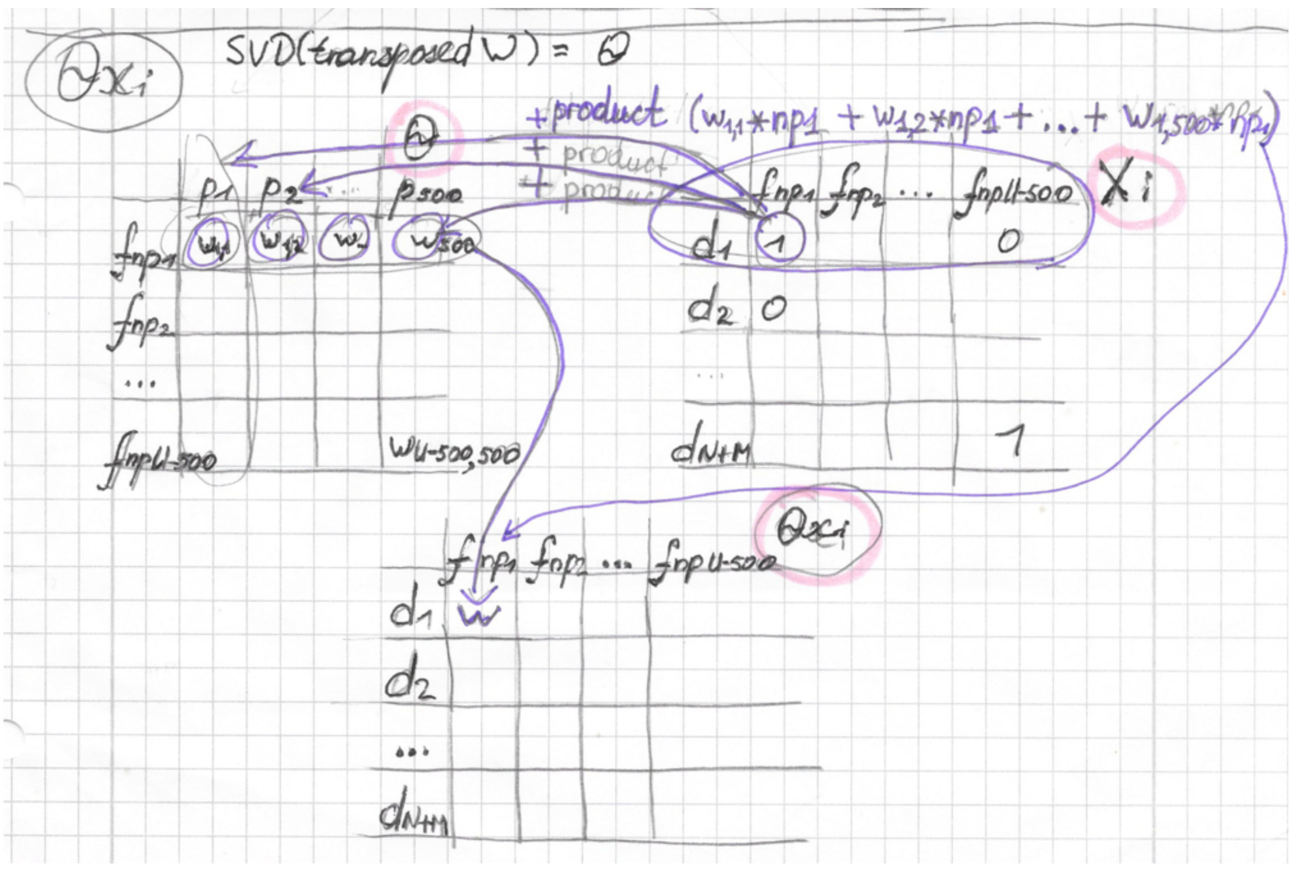
- *We* iterate trough each pivot-feature from 1 to 500 and detect the weights with linear classifier (see the visualization of the pivot predictors in the picture above).
- We have our *W* with predicted weights. The more often pivot and non-pivot have the same values (0 or 1) in the document instances, the higher is the weight, because it is highly likely that they both correspond with each other.

3. Step: SVD(W) (Singular Vector Decomposition)

- In the third step of the algorithm, we transform our weight matrix with singular value decomposition (SVD) for both computational and statistical reasons, as it is useful to compress positive weights and create with it a low-dimensional matrix.
- We call our compressed matrix *θ.*
- Bevor you apply SVD, you need to transpose *W*. This is necessary for dot product (*θ*xi*) in 4. Step

4. Step: train the classifier *f* :

- *xt*: is our non-pivots-matrix;
- *xi*: are our LABALLED data from source transformed into *xt* format;
- we make dot product between *θ* and *xi,* and concatenate *xt* with *θ*xi;*
- now we can train our classifier with labelled data and apply the trained model to the target domain.
- Visualization how to add *θ* representation into training set:



θx_i
SVD(transposed W) = θ
θ
+product ($w_{1,1}$*np_1 + $w_{1,2}$*np_1 + ... + $w_{1,500}$*np_1)
+ product
+ product
p_1 p_2 ... p_{500}
fnp_1 fnp_2 ... fnp_{U-500}
$w_{1,1}$ $w_{1,2}$ w_1 w_{500}
$w_{U-500,500}$
X_i
fnp_1 fnp_2 ... fnp_{U-500}
d_1 d_2 ... d_{N+M}
1 0 0 1
θx_i
fnp_1 fnp_2 ... fnp_{U-500}
d_1 d_2 ... d_{N+M}
w