(In Progress) Noise Removal Methods in Sliding Window Decomposed Decision Tree Model for Spatiotemporal Sequence Prediction

Jiyun "Ivy" Xiao
California Institute of Technology
Pasadena, CA 91126
jxxiao@caltech.edu

Taehwan Kim
California Institute of Technology
Pasadena, CA
taehwan@caltech.edu

Yisong Yue
California Institute of Technology
Pasadena, CA
yyue@caltech.edu

Abstract

Spatiotemporal sequence prediction has become of increasing importance in human motion data and surveillance video data research. High-dimensional spatiotemporal data, however, often contain noise of different amplitudes and sparsity, due to outlier objects tampering data, camera malfunctioning, and many other data corruption that happen in the data collection process. Learning patterns from spatiotemporal data sets that contain random noise is especially challenging since machine learning models fitted to these corrupted data tend to give inaccurate models and render big generalization errors. It is thus important that we either remove the noise before fitting models or build models that have a high tolerance of random noise. Here we combine random noise approaches with sliding window decomposed decision tree models and propose methods that improve upon the naive combination. By testing our methods on a visual speech generation data set, we show that our methods render more accurate models for spatiotemporal sequence prediction and that our methods remove random noise precisely.

1 Introduction

In recent years, high-dimensional data has become ubiquitous in image-related fields such as human motion data, surveillance video data and visual speech data [1,2]. Many of these spatiotemporal data sets, however, are noise-corrupted. In surveillance video data, for example, when the low-resolution images are translated to spatial data and sequenced together into spatiotemporal data, the corrupted pixels in the original images lead to random noise in the resulting data [3].

The general models for learning these data tend to be decision tree or neural network models. While neural network models and decision tree models fit the data set very closely, these close-fitted models also fit the random noise within the training data set, which then causes high generalization errors. The low tolerance of noise in these models makes it important that we remove the random noise in the data set before we train accurate decision tree or neural network models on the data set.

Since the causes and positions of noise in high-dimensional big image sets are quite different, we consider these noise as random noise positioned at unknown locations. One of the most common methods for random noise removal is robust principal component analysis[4,5]. Robust principal

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

component analysis (RPCA) minimizes rank while allowing high tolerance of the random noise in the data sets [5]. RPCA assumes low rank structure and sparse noise, and guarantees exact recovery of the original un-corrupted low rank data matrix.

In this paper, we present two modified-RPCA-decision-tree models, their mathematical basis, algorithms, and experiment results. We show that the both modified-RPCA-decision-tree models are more accurate than the original decision tree model, and that both models recover the original un-corrupted data matrix precisely.

2 Modified RPCA models

We will introduce two modified-RPCA-decision-tree models in this section. We will start by introducing the RPCA method for removing random noise.

Suppose we have a mostly patterned image that contains a irregular object and load it into matrix form. Since the image contains mostly regular patterns, the loaded data matrix will be largely linearly dependent, rendering the matrix low-rank. The irregular object in the image will be loaded into the matrix as random noise. If we want to recover the patterned image without the irregular object, we need to remove the random noise in the image matrix (Figure 1). In the RPCA method, the matrix with unknown random noise is expressed as $M = L + S$, where L is the low-rank matrix to be recovered, and S is the random noise to be removed. The RPCA method ensures exact recovery of L [5].

Figure 1. An example of a patterned image with noise. Left column: original image, where a flag stands in front of an array of windows. Middle column: estimated random noise that roughly fits the flag in the original image, using the RPCA algorithm. Right column: Desired output image where the flag is removed in the picture [5]. This demonstrates that RPCA is an effective algorithm for removing random noise at unknown positions, given that the original un-corrupted data set is low-rank

It is proven [1] that we can recover L exactly through solving

$$min \parallel L \parallel_1 + \parallel S \parallel_*, \text{ such that } L + S = M \tag{1}$$

Using the Lagrangian method, we show that minimizing (1) is equivalent to minimizing

$$\mathcal{L}_{RPCA}(L, S, Y) = \parallel L \parallel_* + \lambda \parallel S \parallel_1 + < Y, M - L - S > + \mu/2 \parallel M - L - S \parallel_F^2 \tag{2}$$

with respect to L,S,and Y. The algorithm for solving (2) is described in Algorithm 1.

Algorithm 1 RPCA algorithm for removing random noise

Require: $S_0 = Y_0 = 0$
1: $\mu \leftarrow n_1 n_2 / 4 ||M||_1$
2: $\delta \leftarrow 10^{-7}$
3: **loop** while $||M - L - S||_F \leq \delta ||M||_F$
4: compute $L_{k+1} = \mathcal{D}_{\mu^{-1}}(M - S_k + \mu^{-1} Y_k)$
5: compute $S_{k+1} = \mathcal{S}_{\lambda\mu^{-1}}(M - L_{k+1} + \mu + Y_k)$
6: compute $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$
7: **return** L

Notations: $\mathcal{S}_\tau[x] = sgn(x)max(|x| - \tau, 0)$. $\mathcal{D}_\tau(X) = U\mathcal{S}_\tau(\Sigma)V^*$, where $X = U\Sigma V^*$ is the singular value decomposition of X.

2.1 Alternate modified-RPCA-decision tree model

Suppose we are just training a decision tree model, then the objective is just minimizing

$$\begin{aligned}\mathcal{L}_(h) &= \Sigma_{(x,y)\in S} l(h(x), y) l(h(x), y) \\ &= \Sigma_{(x,y)\in S} \parallel h(x) - y \parallel_2^2\end{aligned} \tag{3}$$

with respect to h, where h is the decision tree model.

To combine the decision tree model and the RPCA model, we alternately train decision tree for (3) and then train latent variables for (2), until both objective function converge. To be more specific, we train a decision tree first, then use the trained decision tree model to infer latent variables according to RPCA, and then use to updated data matrices to either re-train the decision tree or train a more refined decision tree. See Section 3 for the detailed algorithm (Algorithm 2).

2.2 Combined modified-RPCA-decision-tree model

The alternate modified-RPCA-decision-tree model trains two different objective functions alternately. This could potentially make the model less robust. We thus reformulate the objective function by taking a weighted combination of (2) and (3).

$$\begin{aligned}\mathcal{L}(L, S, Y, h) =& \parallel L \parallel_* + \lambda \parallel S \parallel_1 + < Y, M - S - L > + 1/2 \parallel M - S - L \parallel_2^2 + \gamma \Sigma_{(x,y)\in S} l(h(x), y) \\ =& \parallel L \parallel_* + \lambda \parallel S \parallel_1 + < Y, M - S - L > + 1/2 \parallel M - S - L \parallel_2^2 + \gamma \Sigma_{(x,y)\in S} \parallel h(x) - y \parallel_2^2\end{aligned} \tag{4}$$

Now (1) can be approximated by training a crude decision tree model first and getting L_{ij}, where i is the index of the leaf node the sample is in, and j is the index of the sample within the leaf node, and then setting overall training objective to be

$$\begin{aligned}\mathcal{L}(L, S, Y) =& \parallel L \parallel_* + \lambda \parallel S \parallel_1 + < Y, M - S - L > + 1/2 \parallel M - S - L \parallel_2^2 + \gamma \Sigma_{(x,y)\in S} l(h(x), y) \\ =& \parallel L \parallel_* + \lambda \parallel S \parallel_1 + < Y, M - S - L > + 1/2 \parallel M - S - L \parallel_2^2 \\ &+ \gamma \Sigma_{i\in P} \Sigma_{j\in N_i} \parallel \overline{L_i} - L_{ij} \parallel_2^2 \\ &\approx (1)\end{aligned} \tag{5}$$

where P is the set of all leaf nodes, and N_i the set of all samples with leaf node i, $\forall i \in P$.

The approximation is accurate since $\Sigma_{i\in P} \Sigma_{j\in N_i} \parallel \overline{L_i} - L_{ij} \parallel_2^2$ is the minimized loss of the decision tree model, which is equivalent to the minimum l_2 norm of $\Sigma_{(x,y)\in S} l(h(x), y)$ during the initial training for the less refined decision tree model.

We solve (5) using the gradient descent on $L, S, and Y$. The gradients of S and Y are obvious, so we focus on calculating the gradient of L. We use the proximal gradient algorithm to find gradient of L.

According to the proximal gradient algorithm, we can first decompose (5) into a convex and differentiable (w.r.t. L) part $f(L)$ and a convex and subdifferentiable (w.r.t. L) part $P(L)$, and so

$$\begin{aligned}f(L) &= \lambda \parallel S \parallel_1 + < Y, M - S - L > + 1/2 \parallel M - S - L \parallel_2^2 + \gamma \Sigma_{i\in P} \Sigma_{j\in N_i} \parallel \overline{L_i} - L_{ij} \parallel_2^2 \\ P(L) &= \parallel L \parallel_*\end{aligned} \tag{6}$$

We then calculate the quadratic approximation of (5)

$$Q(L, \tilde{L}) = \frac{\tau}{2} \parallel L - G \parallel_F^2 + P(L) + f(\tilde{L}) - \frac{1}{2\tau} \parallel \nabla f(\tilde{L}) \parallel_F^2, \text{where } G = \tilde{L} - \tau^{-1} \nabla f(\tilde{L}) \tag{7}$$

Therefore, minimizing (5) over L is approximately equivalent to minimizing (6) over L

$$\text{argmin } Q(L, \tilde{L}) = \text{argmin } \frac{\tau}{2} \parallel L - G \parallel_F^2 + P(L) \tag{8}$$

We use singular value thresholding theorem to solve (7). Singular value thresholding theorem states that for each $\tau > 0$, and $Y = \mathbb{R}^{n_1 \times n_2}$, the singular value shrinkage operator obeys $D_\tau(Y) =$ argmin $1/2 \parallel X - Y \parallel_F^2 + \tau \parallel X \parallel_*$ with respect to L, where $X \in \mathbb{R}^{n_1 \times n_2}$ of rank r, $X = U\Sigma V^*, \Sigma = diag(\sigma_i), 1 \leq i \leq r, D_\tau(X) = UD_\tau(\Sigma)V^*, D_\tau(\Sigma) = diag(\sigma_i - \tau_+)$

Applying singular value thresholding theorem to our equation, we get

$$S_{\tau^k}(G^k) = \text{argmin } Q(L, \tilde{L}) = D_{1/\tau}(G) = Udiag(\sigma_i - \frac{1}{\tau_+})V'$$

U, Σ, V are the singular value decomposition of G, of which

$$\begin{aligned} G &= \tilde{L} - \tau^{-1}\nabla f(\tilde{L}) \\ &= \tilde{L} - \tau^{-1}(0 + \frac{\partial < Y, M - S - L >}{\partial \tilde{L}} + \frac{\partial 1/2 \parallel M - S - L \parallel_2^2}{\partial \tilde{L}} + \frac{\partial \gamma \Sigma_{i \in P} \Sigma_{j \in N_i} \parallel \overline{L_i} - L_{ij} \parallel_2^2}{\partial \tilde{L}}) \\ &= \tilde{L} - \tau^{-1}(-Y + M - S - \tilde{L} + C\{2(1 - \frac{1}{N_i})(L_{ij} - \overline{L_i}) - 2\frac{1}{N_i}\Sigma_{k \neq j}(L_{ik} - \overline{L_i})\}_{ij}) \\ &= \tilde{L} - \tau^{-1}(-Y + M - S - \tilde{L} + C\{2(L_{ij} - \overline{L_i})\}_{ij}) \end{aligned} \tag{9}$$

where $C\{a\}_{ij}$ means that the entry corresponding to the ijth node in the matrix C is equal to a.

After we've updated L, we update S and Y using gradient descent. We repeat this loop until the RPCA algorithm converges. We then train a more refined decision tree model based on the recovered low-rank data set. See Section 3 for detailed algorithm (Algorithm 3)

3 Sliding window decomposition in decision tree frameworks

Sliding window decomposition method is based on the observations that spatiotemporal sequences are very context-dependent, and that most spatiotemporal sequences can be approximated by modeling a sufficiently large local neighborhood [1]. By decomposing the problem into a series of overlapping fixed-length prediction problems, sliding window decomposition method maintains the accuracy of the model while making the machine learning problems much more tractable. Below is an illustration of sliding window decomposition method in a decision tree framework applied to a visual speech data set.

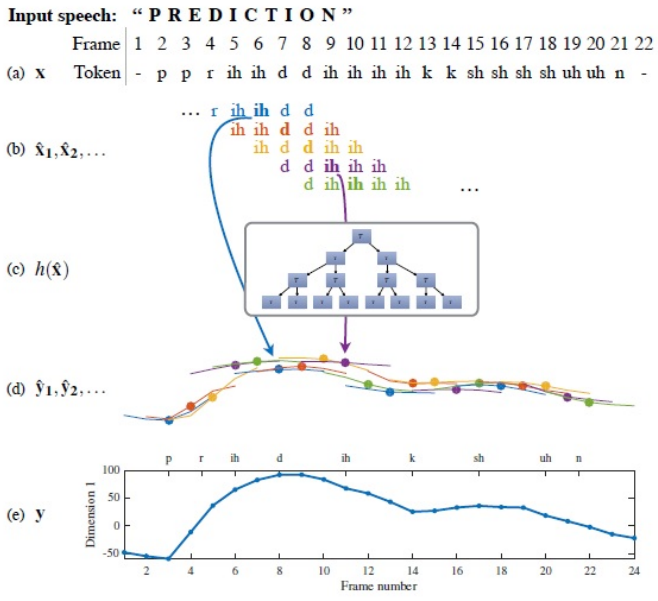

Figure 2. An illustration of the sliding-window-decomposed-decision tree training process. (a)

input: frame-by-frame input from phonetic sequence "prediction"; (b) segment the sequence into input subsequences of length = 5 using sliding window decomposition method; (c) train decision tree models for the data sets consisting of such input subsequences; (d) get output subsequences; (e) blend output subsequences to get the output sequence (only the first dimension is shown for clarity). Our input matrix for the decision tree training process is therefore a matrix of input subsequences that contain same frames[1].

As shown in Figure 1, the same frame can be part of different subsequences and appear multiple times at different locations in the training matrix. In order to make our model physically meaningful, we need to make the value of each frame the same throughout the training process. We address this challenge separately for each of the two modified-RPCA-decision-tree models.

3.1 RPCA-decision tree model

We solve the sliding window decomposition problem in the alternate modified-RPCA-decision-tree model by updating the data matrix so that the frames in different subsequences are of the same value. To be more specific, we first train a decision tree of maxA, a big minimum leaf node size for crude decision tree training. We then loop over each leaf node and apply RPCA to the data matrix within that leaf node, and then substitute the other copies of the nodes within this leaf node with the new value. After updating the entire data matrix, we train a more refined decision tree with a smaller minimum leaf node size called maxB.

Algorithm 2 RPCA-Decision Tree Model

Train Decision Tree with maxA
P = {all leaf nodes}
for each leaf node $p \in P$ **do**
    Apply **Algorithm 1** to the data matrix corresponding to p, get L, S
    for each frame f that contain nonzero entries in S **do**
        for each leaf node $p' \in P - p$ **do**
            If the data matrix corresponding to p' contains frame f, then
            replace it with the corresponding frame in L
Train Decision Tree with maxB

3.2 Reformulated RPCA-decision tree model

We solve the sliding window decomposition problem in the combined modified-RPCA-decision-tree model by changing the structure of the input data for the modified RPCA algorithm. Basically, we train the decision tree with sliding-window-decomposed subsequences, but we blend the subsequences to sequences for removing random noise.

Algorithm 3 Reformulated RPCA Decision Tree Model

Train Decision Tree with maxA
$S_0 = Y_0 = 0$
$\mu \leftarrow n_1 n_2 / 4 ||M||_1$
$\delta \leftarrow 10^{-7}$
loop while $||Y - Z - S||_F \leq \delta ||Y||_F$
    $L^0 = L^{-1} \in domP, t^0 = t^{-1} \geq 1$. For $k = 0, 1, 2, ...$
    while True **do**
        Set $L^k = L^k + \frac{t^{k-1}-1}{t^k}(L^k - L^{k-1})$
        Set $G^k = \tilde{L}^k - (\tau^k)^{-1} \nabla f(\tilde{L}^k)$, where $\tau^k > 0$, and compute $S_{\tau^k}(G^k)$
        Choose a step size $\alpha^k > 0$, and set $L^{k+1} = L^k + \alpha^k (S_{\tau^k}(G^k) - L^k)$
        Choose $t^{k+1} \geq 1$ satisfying $(t^{k+1})^2 - t^{k+1} \leq (t_k)^2$
    $S_{k+1} = \mathcal{S}_{\lambda\mu^{-1}}(M - L_{k+1} + \mu + Y_k)$
    $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$
Decompose the clean data matrix into subsequences, and train decision tree with maxB

4 Experimental Evaluations

We test our models on a visual speech data set using two different test. The first one examines the accuracy of the modified-RPCA-decision-tree models, and the second one examines the effectiveness of random noise removal.

4.1 Data Set

We test our model on a visual speech data set. The data set inputs phonetic inputs of sentences and outputs sequences of actors speaking the sentences (Figure 3). It is of size 220021*150, with each column being a query (0 or 1) about a feature of an image, and each row being a new image. Since people tend to have the same mouth expression when they pronounce the same syllable of sequence of syllables, we expect this visual speech data set to be mostly low-rank. An singular value plotting of the data set also shows that this data set is indeed low-rank.

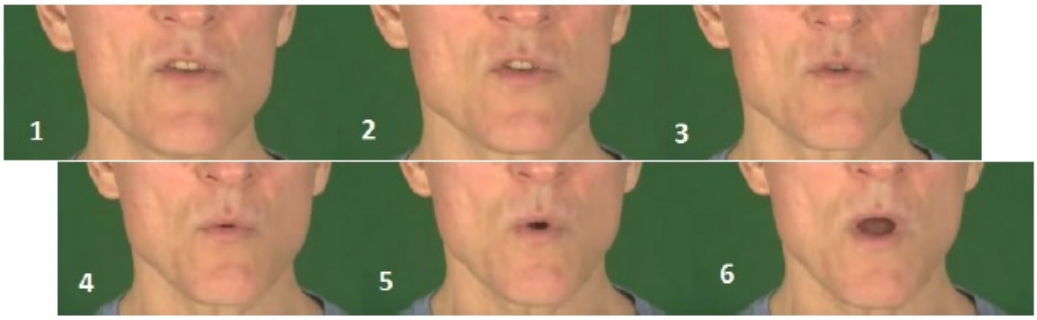

Figure 3. A series of frames of the same actor pronouncing "swa". Each frame image is condensed into 30 parameters using AAM model and then decomposed into overlapping 5-phoneme subsequences using sliding window decomposition. The final input data set is a 220021*150 matrix [1].

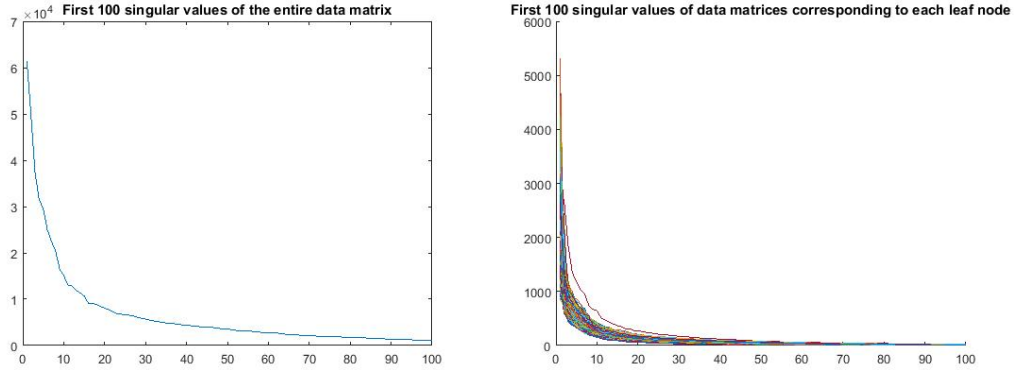

Figure 4. (left) singular values of the entire data matrix. (right) singular values of the data matrix corresponding to each leaf node. Since both are approximate low-rank, we can apply RPCA to both and be guaranteed exact recovery of the clean un-corrupted low rank matrix.

4.2 Experiment Results and Discussion

We compare the testing errors from the alternate modified-RPCA-decision tree model and the original decision tree model from [1] in three different cases: crude decision tree training leaf node size maxA = 2000, 100, and 10. All of the experiment results presented below are generated from data with added Gaussian noise of corrupted ratio = 10%, and models with maxB = 10. Note that minimum leaf node size = 10 is the best performing parameter of the original decision tree models in these three cases.

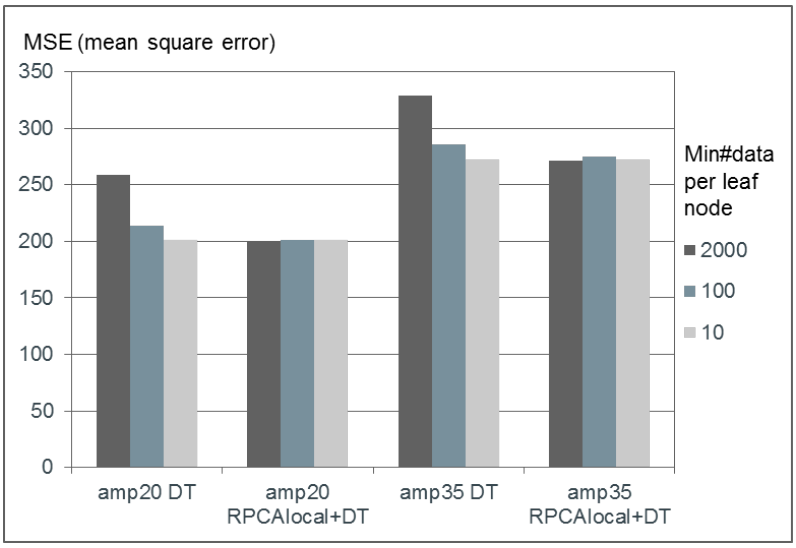

Figure 5 Testing errors of added noise amplitude = 20 data and original decision tree model, added noise amplitude = 20 data and alternate modified-RPCA-decision-tree model, added noise amplitude = 35 data and original decision tree model, added noise amplitude = 35 data and alternate modified-RPCA-decision-tree model. Our modified-RPCA-decision-tree model achieves improvement upon or about the same result compared to the original decision tree model in every case.

We then test the effectiveness of our noise removal method by calculating the 2-norm between the data matrix and the clean data matrix for each loop of the RPCA algorithm. We tested it with both added noise amplitude = 20 and amplitude = 35 cases with part of the entire data matrix, a 2200*30 matrix that preserves most of the data properties. We test it on the modified-RPCA algorithm in the combined modified-RPCA-decision-tree model.

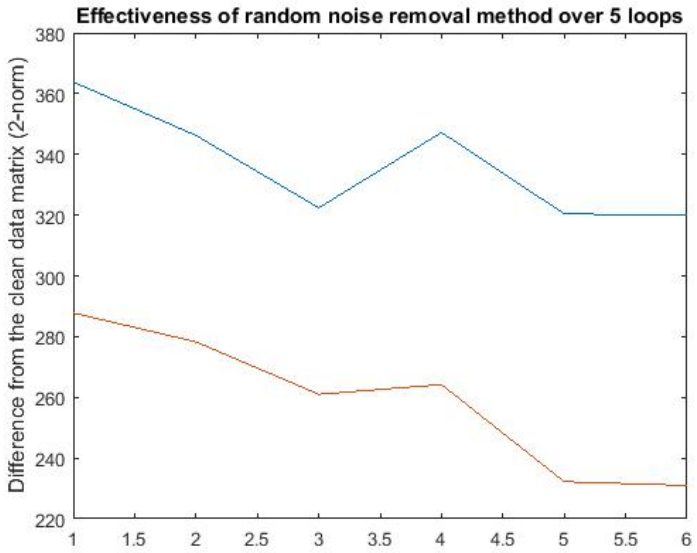

Figure 6 2-norm of the difference between the data matrix and the clean data over 5 loops of modified-RPCA algorithm in the combined modified-RPCA-decision-tree model. The blue line represents the amplitude = 35 results and the orange line represents the amplitude = 20 results. Our results show an overall decreasing trend in difference between the data matrix and the clean matrix in both cases, indicating that our algorithm will eventually converge.

5 Conclusion

We have proposed two modified-RPCA-decision-tree models and presented their detailed mathematical and algorithmic formulation. We validated that our modified-RPCA decision-tree combined models are more accurate than the original decision-tree model without RPCA and that they recover the un-corrupted low rank data matrix precisely by testing our models on a visual speech data set. Directions for future work include generalizing our models to more complicated frameworks including neural network, and proving bounds on the convergence of our models.

6 Acknowledgements

This research was supported by Summer Undergraduate Research Fellowships (SURF) and then continued as the undergraduate thesis program. We thank our sponsors from SURF and our colleagues from Disney Research and Caltech who provided the data set for the research.

References

[1] T.Kim, Y.Yue, S.Taylor, and I.Matthews. A Decision Tree Framework for Spatiotemporal Sequence Prediction. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[2] G.French, M.Fisher, M.Mackiewicz, and C.Needle. Convolutional Neural Networks for Counting Fish in Fisheries Surveillance Video. 2015.

[3] F.Shang, Y.Liu, J.Cheng, and H.Cheng. Robust principal component analysis with missing data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014.

[4] S.Bhalerao, J.Ollitrault, S.Pal, and D.Teaney. Principal component analysis of event-by-event fluctuations. In *Physical review letters*, 2015.

[5] E.Candes, X.Li, Y.Ma, and J. Wright. Robust principal component analysis?. In *Journal of the ACM (JACM)*, 2011.

[6] F.Nelwamondo, S.Mohamed, and T. Marwala. Missing data: A comparison of neural network and expectation maximisation techniques.2007.

[7] H.Daume Iii, J.Langford, and D.Marcu. Search-based structured prediction. In *Machine learning*,2009.

[8] S.Ross, G.Gordon, and D.Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS*, 2011.