Adversarial Autoencoder and Multi-Task Semi-Supervised Learning for Multi-stage Process

Andre Mendes[1(✉)], Julian Togelius[1], and Leandro dos Santos Coelho[2,3]

[1] New York University, New York City, NY, USA
{andre.mendes,julian.togelius}@nyu.edu
[2] Pontifical Catholic University of Parana, Curitiba, PR, Brazil
[3] Federal University of Parana, Curitiba, PR, Brazil
leandro.coelho@pucpr.br

Abstract. In selection processes, decisions follow a sequence of stages. Early stages have more applicants and general information, while later stages have fewer applicants but specific data. This is represented by a dual funnel structure, in which the sample size decreases from one stage to the other while the information increases. Training classifiers for this case is challenging. In the early stages, the information may not contain distinct patterns to learn, causing underfitting. In later stages, applicants have been filtered out and the small sample can cause overfitting. We redesign the multi-stage problem to address both cases by combining adversarial autoencoders (AAE) and multi-task semi-supervised learning (MTSSL) to train an end-to-end neural network for all stages together. The AAE learns the representation of the data and performs data imputation in missing values. The generated dataset is fed to an MTSSL mechanism that trains all stages together, encouraging related tasks to contribute to each other using a temporal regularization structure. Using real-world data, we show that our approach outperforms other state-of-the-art methods with a gain of 4x over the standard case and a 12% improvement over the second-best method.

Keywords: Multi-stage · Multi-task · Autoencoder

1 Introduction

In many applications including network intrusion detection [15] and medical diagnosis [1], decision systems are composed of an ordered sequence of stages, which can be referred to as a multi-stage process. For selection processes (such as hiring or student intake), for instance, the applicants submit general information in the initial stages, such as resumes. The evaluator screens trough the resumes and selects applicants to move on to the next round. In each following stage, applicants are filtered out until the final pool is selected. In terms of information, the initial stages have general data about the applicants and for each subsequent

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12085, pp. 3–16, 2020.
https://doi.org/10.1007/978-3-030-47436-2_1

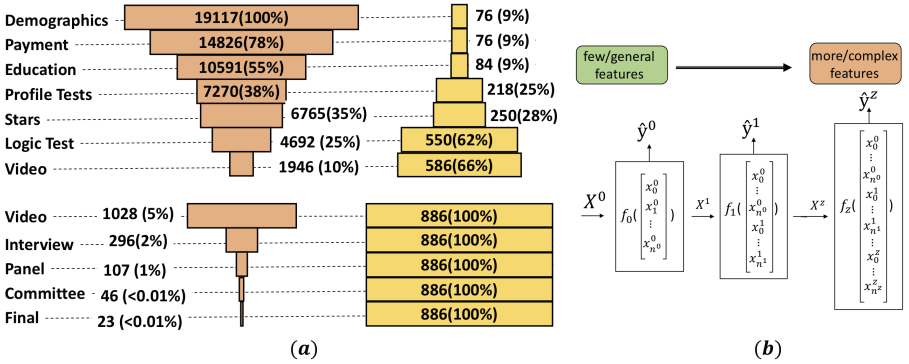

Fig. 1. Dual funnel structure (a); multi-stage architecture (b). In (a), the left funnel shows the number of applicants decreasing, whereas the right funnel shows the data (in terms of variables) increasing. In (b), for every stage $\{s\}_0^S$, a classifier f_s make decisions $\hat{Y}^s$ using features X^s.

round, more and specific information is gathered. As the process continues, the cost to acquire and evaluate new data increases, which is an incentive to decrease the pool of applicants. Hence, in the final stages, the evaluator has a smaller pool with much more information about each applicant.

Training classifiers for a multi-stage process can be challenging because of the dual funnel structure as shown in Fig. 1(*a*). During the initial stages, the number of applicants decreases whereas the data about them increases. Therefore, the dataset grows in dimensionality but decreases in terms of sample size. Classifiers trained in initial stages have sufficiently large samples to generalize, but the available features might not contain enough information to differentiate applicants, causing high bias and underfitting. In the final stages, there is more information for each applicant, but the sample size is reduced significantly, causing classifiers to suffer from high variance and overfitting. To address this problem, we redesign the multi-stage problem and present a framework with two components.

In the first part of our framework, we use an *adversarial autoencoder* (AAE) [19] that learns the data representation and is able to generate synthetic data for data imputation in missing values. This component makes possible for our framework to fill the data for an applicant in all the stages that he has not reached. Therefore, we can generate a complete dataset with data for all applicants in all stages.

In the second part of our framework, we use multi-task learning to train a single classifier that can learn different tasks together. We introduce a temporal regularization structure so that related tasks can share information, and we adopt a semi-supervised approach to handle the newly generated samples from AAE, which don't have labels. This results in the a**D**versarial auto**E**n**C**oder and mult**I**-task **S**em**I**-super**V**ised l**E**arning (DECISIVE) framework. The main contributions of this paper are:

1. We adapted an adversarial auto-encoder to perform data imputation and generate a complete dataset to be shared in different stages.
2. We redesign the multi-stage problem and present a temporal multi-task semi-supervised framework that allows knowledge from all stages to be shared.
3. The effectiveness of the proposed model is demonstrated by extensive longitudinal experiments on real data from 3 different selection processes. Our model is able to outperform other single-task and multi-task frameworks particularly in the later stages where the sample size is significantly reduced.

2 Related Work

Our method builds on three subfields of machine learning, namely data imputation [3], multi-task learning [20] and multi-stage or decision cascades [15,17].

Data Imputation with Autoencoders - Many methods for data imputation have been proposed, ranging from simple column average to complex imputations based on statistical and machine learning models. Such methods can be categorized in discriminative [14], or generative [16]. Missing data is a special case of noisy input and deep architectures such as *denoising autoencoders* (DAE) [16] have performed well due to their capability to automatically learn latent representations. The work presented in [5] uses an overcomplete DAE as a base model to create a multiple imputation framework, which simulates multiple predictions by initializing the model with random weights at each run. Our AAE component is based on the framework proposed in [19], which explores the generative adversarial network (GAN) [6] approach to create: a generator to accurately impute missing data; a discriminator to distinguish between observed and imputed components; and a hint mechanism to help generate samples according to the true underlying data distribution.

Multi-Task and Semi-Supervised Learning - The goal of multi-task learning (MTL) is to learn multiple related tasks simultaneously so that knowledge obtained from each task can be re-used by the others. This can increase the sample size for each task, making MTL beneficial for tasks with small training sample. MTL has been applied in different approaches including neural nets (NN) and kernel methods [1,7]. More recent methods have explored the application of MTL to deep neural nets (DNN) [11]. The regularization parameters in MTL control how information is shared between tasks and prevents overfitting. The framework proposed in [7] enables information to be selectively shared across tasks by placing a structure constrain on the learned weights. Our framework builds on [21], in which temporal information is encoded using regularization terms. MTL can also be combined with semi-supervised learning (SSL) to create classifiers coupled by a joint prior distribution over the parameters of all classifiers [4].

Multi-stage Classification - Multi-stage and cascade classifiers share many similarities, however, an important difference between cascade [17] and multi-stage can be defined as the system architecture. Detection cascades make partial decisions, delaying a positive decision until the final stage. In contrast,

multi-stage classifiers shown in Fig. 1(*b*), can deal with multi-class problems and can make classification decisions at any stage [15]. The approach proposed in [12] explores the connection between deep models and multi-stage classifiers such that classifiers can be jointly optimized and they can cooperate across the stages. This structure is similar to ours, but in our case, the algorithm only has access to the original information in a specific stage.

3 Problem Statement

Let's define $s = \{0, 1, ...S\}$ as a single stage in a multi-stage process. Every stage has a dataset with training examples $\{x_i^s, y_i^s\}_{i=0}^{m^s}$, where m^s refers to the number of samples. The number of features is given by n^s, the features are given by $X^s \in \mathbb{R}^{m^s \times n^s}$ and the labels by $Y^s \in \mathbb{R}^{m^s \times 1}$. Let's also define $A \in \mathbb{R}^{m \times 1}$ as the vector of applicants, where m represents the total number and $a \in A$ represents a single applicant. The feature vector for a single applicant a in stage s are given by the vector $x_a^s \in \mathbb{R}^{1 \times n^s}$. A prediction matrix $\hat{P} \in \mathbb{R}^{m \times S}$ can be defined, where $p_a \in \mathbb{R}^{1 \times S}$ is the prediction vector for an applicant a in all stages, and $p_a^s \in [-1, 1]$ represents the prediction for a single stage s.

Underfitting in Earlier Stages - In each stage s, only features x^s up to that stage are available. Therefore, a prediction for an applicant with data up to s is given by $p_a^i = f(x_a^s)$, $i = \{s, s+1, ..., S\}$, where $f(\cdot)$ is a classification function. For example, for an applicant with information in $s = 0$, all predictions will be made using x_a^0.

Since the features in early stages are more general and less discriminative, models trained in these stages have poor performance predicting the applicant's future in the process. A method to address this problem could incorporate future features in the early stages. More specifically, a data imputation process can be used to fill missing information and generate a complete dataset for all stages. In other words, $\hat{X} = g(X^s)$, where X^s is the input features in stage s and $g(\cdot)$ is a data imputation function.

Overfitting in Later Stages - For each new stage, new data is received while the number of samples decrease, which means $X^{s+1} \neq X^s$, $m^{s+1} < m^s$ and $n^{s+1} > n^s$. As m^s gets significantly smaller in absolute value and in comparison to n^s, classifiers trained on the specific stage data tend to overfit. One possible way to address this problem is to use the generated complete dataset $\hat{X}$. Since any stage X^s can be mapped to $\hat{X}$, more training samples can be used to train classifiers in later stages. However, $\hat{X}^{s+1} = g(X^s)$ only generate new samples but no labels, since the applicants in s were not evaluated in $s+1$. Hence, $Y^{s+1} \neq Y^s$ and $m^{s+1} < m^s$ for the label matrix. In this case, semi-supervised learning can be applied so that labeled and unlabelled data are combined to create a better classifier than by just using the labeled data.

Finally, with S stages, the traditional way would be to construct S classifiers to predict the outcomes in each stage. However, we believe that these tasks in each stage are related and they could be combined to share knowledge during training. This problem can be addressed by an MTL that seeks to improve the generalization performance of multiple related tasks by learning them jointly.

4 Methods

Our method is based on two components: data imputation using adversarial autoencoders (AAE) and multi-task semi-supervised learning (MTSSL). Here we explain each of these components as well as the combination of them to form our approach.

4.1 Data Imputation Using Adversarial Autoencoders (AAE)

In each stage, we have a combination of numerical and categorical features (encoded using one-hot encode). For data imputation, our method is based on [19] and it is shown in Fig. 2. We use an adversarial approach in which the generator G receives as input features $X \in \mathbb{R}^{m \times n}$, a binary mask B indicating the positions of the missing data in X, and a source of noise. We create the dataset $\tilde{X}$ by filling the missing positions in X with the random noise. The goal of G is to generate the data $\hat{X}$ with values as close as possible to the original values X. We also have a discriminator D that receives $\hat{X}$ and tries to guess if each variable value is either original or imputed.

Additionally, a hint mechanism is added by using a random variable H to depend on B. For each (imputed) sample $(\hat{x}, b)$, we draw h according to the distribution $H|B = b$, and h is passed as an additional input to the D. The hint H provides information about B so that we can reduce the number of optimal distributions with respect to D that G could reproduce. Therefore, the discriminator tries to predict a binary mask $\hat{B} = D(\hat{X}, H)$ that is as close as possible to B. We define the cross-entropy loss function as

$$\mathcal{L}x(a, b) = \sum_{i=1}^{d} a_i \log b_i + (1 - a_i) \log(1 - b_i), \tag{1}$$

In the adversarial approach, D maximizes the probability of correctly predicting B while G minimizes the probability of D predicting B, which is given by:

$$\min_G \max_D \ \mathbb{E}[\mathcal{L}x(B, \hat{B})], \tag{2}$$

where G influences the loss by generating $\hat{X}$ in the term $\hat{B} = D(\hat{X}, H)$. To solve this problem, we first optimize the discriminator D with a fixed generator G. For a mini-batch k_D, the discriminator D is trained to optimize

$$\min_D \sum_{j \in k_D} \mathcal{L}x_D(B_j, \hat{B}_j), \tag{3}$$

Second, we optimize the generator G with the newly updated discriminator D using

$$\mathcal{L}_G(b, \hat{b}, z) = - \sum_{i:z_i=0} (1 - b_i) \log \hat{b}_i, \tag{4}$$

where $z \in Z$ and $Z \in \{0,1\}^n$ is a random variable defined by first sampling k from $\{1,..n\}$ uniformly at random. The reconstruction error for the non-missing values is

$$\mathcal{L}_{rec}(x,\hat{x}) = \sum_{i=1}^{n} b_i rec(x_i,\hat{x}_i), \tag{5}$$

$$rec(x_i,\hat{x}_i) = \begin{cases} (\hat{x}_i - x_i)^2 \text{ if } x_i \text{ is continuous} \\ -x_i \log \hat{x}_i \text{ if } x_i \text{ is binary} \end{cases} \tag{6}$$

The final equation for G in a mini-batch k_G and a hyperparameter α is given by

$$\min_G \sum_{j=1}^{k_G} \mathcal{L}_G(b_j,\hat{b}_j,z_j)) + \alpha(\mathcal{L}_{rec}(\tilde{x_j},\hat{x}_j)), \tag{7}$$

4.2 Multi-Task Semi-Supervised Learning (MTSSL)

In this component, we use a SSL approach to create a model that can use both the labeled and unlabeled data together for model training. We combine SSL and MTL, so that different tasks can be learned simultaneously in a joint framework. Let's define the input feature $X \in \mathbb{R}^{m \times n}$ with labeled samples, $X_L \in \mathbb{R}^{m_L \times n}$ with corresponding labels $Y \in \mathbb{R}^{m_L \times 1}$ and unlabeled samples $X_U \in \mathbb{R}^{m_U \times n}$. Let's define a task t as the task to predict the label for a applicant in a given stage. For S stages, we have T tasks, hence $T = S$. For supervised learning in a task t, we use the cross-entropy loss

$$\mathcal{L}_L(X_L^t, Y^t) = -\sum_{i=1}^{m_L^t} y_i^t \log \hat{y}_i^t + (1 - y_i^t) \log(1 - \hat{y}_i^t). \tag{8}$$

To achieve semi-supervised learning, we rely on the common assumption that if two feature vectors x_i and x_j are close in a weighted graph, their predictions $f(x_i)$ and $f(x_j)$ should be similar [4]. Therefore, we can use a graph regularization term that depends on the affinity similarity matrix O, where the affinity similarity between x_i and x_j is given by

$$o_{ij} = \begin{cases} \exp(-\frac{||x_i - x_j||^2}{\sigma_i \sigma_j}), & x_i \in N_K(x_j) \text{ or } x_j \in N_K(x_i), \\ 0, & otherwise, \end{cases} \tag{9}$$

where $N_K(x_i)$ denotes the K nearest neighborhood set of x_i. The tuning parameters σ_i and σ_j can be set as the standard deviation of the related K nearest neighbor set. By using the affinity matrix, we can calculate the semi-supervised term as:

$$\mathcal{L}_U(X_U^t) = \gamma_1 \sum_{i,j \in t}^{m_U^t} o_{ij}^t ||f^t(x_i^t) - f^t(x_j^t)||_F^2. \tag{10}$$

As shown in [4], Eq. 10 can be simplified in a closed form solution as

$$\mathcal{L}_U(X_U^t) = \gamma_1 tr(f^t \Delta^t (f^t)^T), \tag{11}$$

where $\gamma_1 0$ is a model tuning hyperparameter; $\Delta^t = D^t - O^t$ is the graph Laplacian Matrix and D^t is a diagonal matrix with $d_{ii} = \sum_j o_{ij}^t$ and $t = \{0, 1, ..., T\}$.

For regularization, we introduce a temporal structure to encourage sequential tasks to share knowledge. This is achieved with a graph regularization similar to the affinity matrix but applied to tasks. An edge $r \in R$ can be defined as

$$r_{ij} = \begin{cases} 1, & j = i + 1 \ or \ i = j + 1, \\ 0, & otherwise, \end{cases} \tag{12}$$

Putting everything together, the multi-task semi-supervised loss function is given by

$$\min_W \sum_{t=1}^{T} \sum_{j=1}^{m_L^t} \mathcal{L}_L(y_j^t, W^T x_j^t) + \mathcal{L}_U(X_U^t) + \lambda_1 ||W||_F^2 + \lambda_2 ||WR||_F^2 + \lambda_3 ||W||_{2,1}, \tag{13}$$

where x_j^t denotes sample j of the *t-th* task, y_j^t denotes its corresponding label, X_U^t is the unlabeled dataset, W are the model parameters, λ_1 controls the l_2-norm penalty to prevent overfitting, λ_2 is the regularization parameter for temporal smoothness and λ_3 controls group sparsity for joint feature selection.

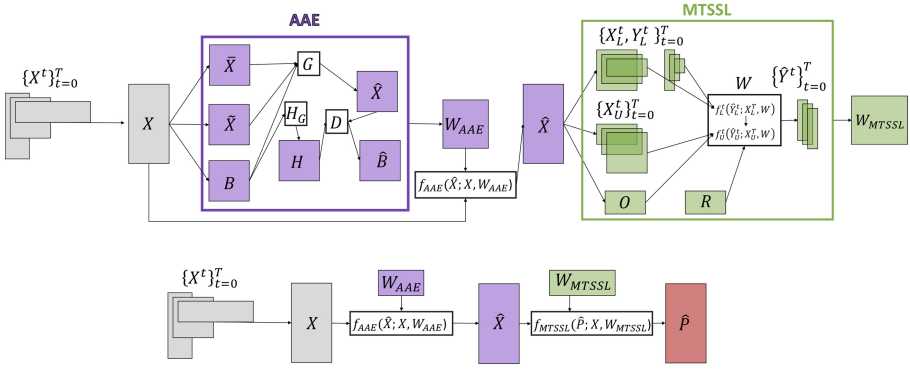

Fig. 2. DECISIVE framework. On top-left, input features $\{X^t\}_{t=0}^T$ are concatenated to form the general matrix X, which is fed to the Adversarial Autoencoder (AAE) component. Inside AAE, A generator G, a discriminator D and a hint generator H_G are trained to learn the representation of the data and produce the weights W_{AAE} for data imputation. On top-right, AAE produces the complete dataset $\hat{X}$, which is fed to the multi-task semi-supervised (MTSSL) component. MTSSL uses labeled $\{X_L^t, Y_L^t\}_{t=0}^T$ and unlabeled $\{X_U^t\}_{t=0}^T$ datasets for each task as well as an affinity matrix O to train the classifier for all tasks. The weights W are regularized using a temporal graph R that enforces temporal smoothness. On the bottom, the prediction flow uses W_{AAE} to generate the complete dataset $\hat{X}$ and W_{MTSSL} to generate the prediction matrix $\hat{P}$.

4.3 Adversarial Autoencoder and Multi-Task Semi-Supervised Learning

Figure 2 shows our entire method. We first use the AAE component to generate $\hat{X}$, which will be the input in the MTSSL component to obtain the predictions. This framework allows us to (1) create a dataset common to all stages that can be used for other tasks and (2) train a single classifier for all stages promoting contribution among correlated tasks and better generalization using labeled and unlabeled data.

5 Experiments

To demonstrate the application of our method in a real-world setting, we perform experiments using 3 datasets from 3 different multi-stage selection processes.

Datasets - For privacy requirements, we are going to refer to the companies with indexes such that C_1 refers to Company 1. The companies have a similar process but they have distinct goals: C_1 is an organization that selects students for a fellowship; C_2 is a big retail company and its process focus on their recent-grad hire program; C_3 is a governmental agency that select applicants to work in the public administration.

Each dataset contains sub-datasets. For example, in C_1, the process happens annually and we have data for three years, $year^1_{C1}$, $year^2_{C1}$, $year^3_{C1}$. The dual funnel structure from this process is shown in Fig. 1. The process in C_2 happens every semester and data for 6 semesters is available, on average, 13000 applicants start the process and 300 are selected. The process in C_3 happens annually and data from 2 years are available. In this case, 5000 applicants start the process and 35 are selected.

Each stage in the process contains its own set of variables. For example, in the stage *Demographics*, information about state and city is collected. Therefore, we refer to *Demographics* features for those collected in the *Demographics* stage. The data collected in each process is very similar in stages such as *Demographics* and *Education*, both in the form of content and structure. For stages with open questions such as *Video*, each process has its own specific questions (See Table 1 for details). Additionally, in all datasets, the *Video* stage marks the last part where data is collected automatically.

Feature Preparation - Early stages have structured data in a tabular format. Categorical variables are converted to numerical values using a standard one-hot encode transformation. In the later stages, such as video, the speech is extracted and the data is used as a text. To convert this data to numerical values, we create word embeddings using Word2Vec [9], where we assign high-dimensional vectors (embeddings) to words in a text corpus but preserving their syntactic and semantic relationships.

Validation and Performance Metrics - We perform longitudinal experiments, in which we use a previous year as a training and test set and the following year as a validation set. For C_1 for example, we split the dataset from $year^1_{C1}$ in train and test, find the best model and validate its results using the dataset from

Table 1. Stages in the multi-stage selection process

Stages	Company 1 (C_1)	Company 2 (C_2)	Company 3 (C_3)
Demo	Provide country, state, city	Same as C_1	Same as C_1
Payment	Pay application fee	Not applicable	Not applicable
Education	Provide university, major and extra activities	Same as C_1	Same as C_1
Profile test	Online tests to measure profile characteristics such as ambition and interests	Online tests to measure big 5 characteristics	Same as C_1
Experience	Write on professional experience using the model (S: situation, T: task, A: action, R: result)	Write about important professional experience	Same as C_1
Logic test	Perform online tests to map levels in problem-solving involving logic puzzles	Same objective as C_1 but with specific test for C_2	Same objective as C_1 but with specific test for C_3
Video submission	2-min, explaining why they deserve the fellowship	5-min, making a case to be selected for the position	1-min, explaining a problem that motivates the applicant
Video evaluation	Applicants are evaluated based on their entire profile submitted	Same as C_1 but with different criteria	Same as C_1 but with different criteria
Interview	1-on-1 interview to clarify questions about the applicant's profile	Same as C_1 but with different criteria	Same as C_1 but with different criteria
Panel	Former fellows interview 5 to 6 applicants at the same time in a group discussion	Managers interview 4 applicants in a group discussion	Not applicable
Committee	Senior fellows and selection team select applicants to move to the final step	Not applicable	Not applicable
Final	Applicants are interviewed by the board of the company	Applicants are interviewed by a group of directors	Applicants are interviewed by a group of managers

$year^2_{C1}$. The model is trained in $year^1_{C1}$ and has never seen any data in $year^2_{C1}$. We repeat the process for all other years. Finally, we also combine the datasets from $year^1_{C1}$ and $year^2_{C1}$ and validate the results in $year^3_{C1}$, which results in 4 groups. For the train and test split, we also perform 10-fold cross-validation resulting in a total of 40 runs. In C_2, we obtain 33 groups (330 runs) and for C_3 we have only 1 group (10 runs). In all experiments, we compare the models in terms of F1-score for the positive class, which balances precision and recall specific for the applicants that are selected in each stage.

Benchmark Methods - We compare the results with other established methods such as: *Support Vector Machines* (SVM) [2] with $C = 0.1$; and neural networks (NN) with dropout regularization, $p = 0.5$ [13]. We apply them in the standard setting, i.e. without AAE (N), Single Task Learning (STL) and Supervised Learning (SL), N-STL-SL. In our second setting, we add data imputation and run experiments with semi-supervised learning, AAE-STL-SSL. For this setting we use: *Laplacian Support Vector Machines* (LapSVM) [8] with Gaussian kernel, $\sigma = 9.4, nn = 10, p = 2, \gamma_A = 10^{-4}$, $\gamma_I = 1$; and DNN with *Virtual Adversarial Training (VAT)* [10] with $K = 1, \epsilon = 2$, $\alpha = 1$. For NN methods, we

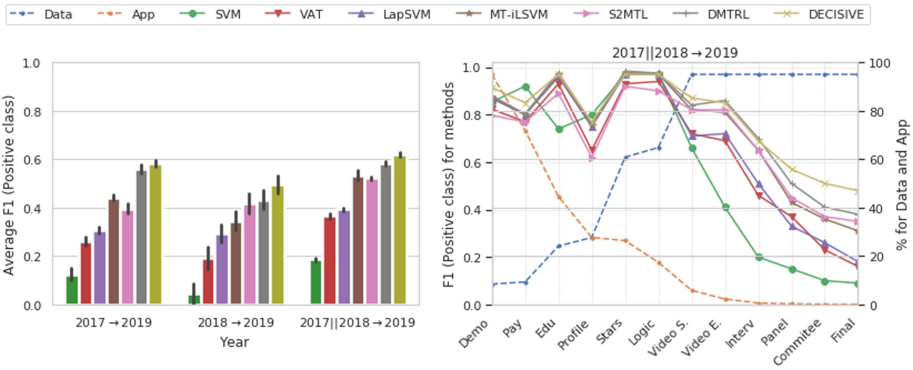

Fig. 3. LEFT - Averaged results for C_1 in later stages (Video E. to Final); RIGHT - Results for a single year in all stages for C_1. On the right, for each new stage, the number of applicants drops while more data is obtained. Methods perform better while the number of samples is relatively high but the performance worsens drastically in the later stages (Video E. to Final).

use dense NNs with fully connected layers and the structure is chosen such that the number of parameters is similar in all settings.

Our third setting uses the MTL component in the SL case, hence AAE-MTL-SL. We use: *Infinite Latent SVM* (MT-iLSVM) [22] with $\alpha = 0.5$, $C = 1$, $\sigma_{m0}^2 = 1$; and *Deep Multi-Task Representation Learning with tensor factorization* (DMTRL) with Tucker method [18]. Finally, in the fourth setting, AAE-MTL-SSL, we use DMTRL in the SSL setting and *Semi-Supervised Multi-task Learning* (S2MTL) [4] with $\lambda_1 = 10$, $\lambda_2 = 10$, $k = 4$. For our approach, DECISIVE, we use $\lambda_1 = 0.1$, $\lambda_2 = 1$, $\lambda_3 = 10$ and $\gamma_1 = 10$. All hyperparameters are found using cross-validation and the performance is robust for values chosen in the interval $[10^{-2}, 10^{-1}, 1, 10, 100]$.

6 Results

In this section we present the results obtained in all experiments, which can be seen in Fig. 3 for C_1 and in Table 2 and Table 3 for C_2 and C_3, respectively. *Data* refers to the number of features in each stage compared to the final dataset. *App* refers to the number of applicants in each stage compared to the number in the first stage.

6.1 General Results Across Companies

We investigate the general results in all experiments. SVM achieved the best result in the N-STL-SL (to save space we omit NN), however, this group had the worst performance mostly due to overfitting, since each classifier is trained individually and knowledge across stages is not shared. The second-best group

Table 2. Average results for each stage in terms of F1 for C_2.

				N-STL-SL	AAE-STL-SSL		AAE-MTL-SL	AAA-MTL-SSL			
		Data (%)	App (%)	SVM	VAT	LapSVM	MT-iLSVM	S2MTL	DMTRL	DECISIVE	MEAN
Initial	Demo	8.6	100	0.75	0.74	0.79	0.71	0.7	**0.84**	**0.8**	0.76
	Edu.	23.7	62	0.6	0.75	**0.84**	0.82	0.71	0.81	**0.85**	0.77
	Profile	28.2	54	**0.78**	0.55	0.67	0.71	0.67	0.7	**0.72**	0.69
	Star	62.1	38	**0.87**	0.81	0.79	0.79	0.78	**0.9**	**0.87**	0.83
	Logic	66.1	31	**0.91**	0.78	0.93	0.9	0.88	**0.91**	0.88	0.88
	Video S.	100.0	15	0.46	**0.68**	0.59	0.7	0.62	**0.7**	0.67	0.63
	MEAN			0.73	0.72	0.77	0.77	0.73	**0.81**	**0.80**	
Late	Video E.	100.0	12	0.17	0.48	0.55	0.58	0.53	**0.65**	**0.64**	0.51
	Interview	100.0	6	0.13	0.38	0.43	0.63	0.61	**0.62**	**0.64**	0.49
	Panel	100.0	4	0.14	0.31	0.25	0.35	0.41	**0.43**	**0.55**	0.35
	Final	100.0	1.5	0.08	0.12	0.12	0.25	0.3	**0.29**	**0.47**	0.23
	MEAN			0.13	0.32	0.34	0.45	0.46	**0.50**	**0.58**	

is VAT and LapSVM in AAE-STL-SSL. Classifiers in this group are still trained individually, but the additions of data imputation and unsupervised data help the classifier to better generalize in the later stages. SVM methods (LapSVM) have better performance than NN (VAT) especially when the sample size is relatively small.

When MTL is introduced in the third setting, AAE-MTL-SL, results are improved since knowledge is shared in all tasks. The parameters for the multi-task classifier are updated based on the loss for all predictions which helps mitigate the challenge of small sample size in the later stages (Video E. to Final). In this setting, the NN-based method (DMTRL), can perform similar to the SVM-based method (iLSVM), since the NN can learn more complicated patterns and decision boundaries without overfitting as in previous cases. However, the best results for DMTRL are in the SSL case.

The best performing methods are in the AAE-MTL-SSL setting. DMTLR performs slightly better than the S2MTL method, in which, knowledge is shared at the feature level. In general, our method (DECISIVE) can outperform the other methods and the difference is higher in later stages (see Sect. 6.2). Comparing to DMTRL and S2MTL, the temporal regularization introduced in our method enforces tasks close together in time to contribute more with each other. This allows our model to share knowledge among all tasks in the multi-task framework, but still, preserve some of the temporal components of the process.

6.2 Company Results

Figure 3-left shows that the results among methods are consistent for C_1 as we vary the training data. Interestingly, predicting the data in $year^3_{C_1}$ using $year^1_{C_1}$ is better than using $year^2_{C_1}$. This shows that the profile for applicants changes from year to year and the applicants approved in $year^3_{C_1}$ are more similar to the ones approved in $year^1_{C_1}$. Therefore, it is important to have data from different years to improve generalization. When we combine data from $year^1_{C_1}$ and $year^2_{C_1}$, the results improved due to the increase in sample size and the combination

Table 3. Average results for each stage in terms of F1 for C_3.

				N-STL-SL	AAE-STL-SL		AAE-MTL-SL	AAE-MTL-SSL			
		Data (%)	App (%)	SVM	VAT	LapSVM	MT-iLSVM	S2MTL	DMTRL	DECISIVE	MEAN
Initial	Demo.	9.16	100	0.8	0.75	0.79	0.78	0.72	**0.83**	**0.87**	0.79
	Edu.	18.07	40	0.69	0.84	0.88	0.88	0.79	**0.9**	**0.89**	0.84
	Profile	24.1	20	**0.75**	0.6	0.66	0.7	0.57	0.68	**0.76**	0.67
	Star	60.24	10	0.87	0.88	0.87	**0.92**	0.86	0.91	**0.92**	0.89
	Logic	63.86	6	0.92	0.87	0.9	0.91	0.83	**0.93**	**0.95**	0.90
	Video S.	100	4	0.56	0.66	0.64	0.77	0.74	**0.79**	**0.82**	0.71
	MEAN			0.77	0.77	0.79	0.83	0.75	0.84	0.87	
Late	Video E.	100	2	0.41	0.41	0.52	0.51	0.52	**0.54**	**0.56**	0.50
	Interview	100	>0.5	0.2	0.2	0.23	0.37	0.38	**0.44**	**0.47**	0.33
	Final	100	>0.1	0.09	0.08	0.08	0.11	0.09	**0.14**	**0.22**	0.12
	MEAN			0.23	0.23	0.28	0.33	0.33	0.37	0.42	

of profiles from different years. In terms of stages, it is clear from the right figure that algorithms can perform well while there is enough data to generalize. However, in later stages, there is a drop in performance caused by the small sample size. Algorithms based on AAE-MTL-SSL perform significantly better than others. They achieved a gain of 4x compared to the best standard case (SVM) for later stages. Additionally, our method outperforms the second best with a 12% gain.

Results for C_2 are shown in Table 2. This process contains more longitudinal data (6 semesters), which makes our model more robust to changes across years. Additionally, the number of applicants is more evenly distributed and more applicants reach the final stages, which causes the average performance to be more similar across all stages. In other words, the drop in performance is less steep for later stages compared to C_1. We also see that learning from unsupervised samples is less impactful, as methods from AEE-MTL-SL (MT-iLSVM) have similar performance to methods in AEE-MTL-SSL (DMTRL, DECISIVE). Our method outperforms the other methods in this case (16% gain), which shows the importance of the multi-task and regularization in our structure. Compared to SVM, the gain is about 3.46x. Both gains related to the later stages.

Table 3 shows the results for C_3. This process has an overall smaller sample size and only two years are available. We reason that the methods could overfit the training set and not generalize so well to the validation set. Nevertheless, the algorithms in the AAE-MTL-SSL setting still have the best performance, but the difference from the standard case is smaller when compared to the other experiments in later stages (80% gain). Our approach can outperform the other methods with a 13% gain over DMTRL.

7 Conclusion

We presented a framework that combines adversarial autoencoders (AAE) and multi-task semi-supervised learning (MTSSL) to train an end-to-end neural network for all stages of a selection problem. We showed that the AAE makes

it possible to create a complete dataset using data imputation, which allows downstream models to be trained in SL and SSL settings. We also introduced a temporal regularization on the model to use information from different stages but still conserve the temporal structure of the process. By combining MTL and SSL, our method can outperform other STL and SL methods. Our validation includes real-world data and our method is able to achieve a gain of 4x over the standard case and a 12% improvement over the second best method. For future research, we want to introduce interpretability techniques to understand the profiles learned by the model and investigate the effect of bias. We also want to apply the framework to other multi-stage processes in different fields.

References

1. Caruana, R.: Multitask learning. Mach. Learn. **28**(1), 41–75 (1997)
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. (TIST) **2**(3), 27 (2011)
3. Chen, P.: Optimization algorithms on subspaces: revisiting missing data problem in low-rank matrix. Int. J. Comput. Vis. **80**(1), 125–142 (2008)
4. Cong, Y., Sun, G., Liu, J., Yu, H., Luo, J.: User attribute discovery with missing labels. Pattern Recognit. **73**, 33–46 (2018)
5. Gondara, L., Wang, K.: MIDA: multiple imputation using denoising autoencoders. In: Phung, D., Tseng, V.S., Webb, G.I., Ho, B., Ganji, M., Rashidi, L. (eds.) PAKDD 2018. LNCS (LNAI), vol. 10939, pp. 260–272. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93040-4_21
6. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
7. Kumar, A., Daumé III, H.: Learning task grouping and overlap in multi-task learning. In: Proceedings of the 29th International Conference on International Conference on Machine Learning, pp. 1723–1730. Omnipress (2012)
8. Melacci, S., Belkin, M.: Laplacian support vector machines trained in the primal. J. Mach. Learn. Res. **12**(Mar), 1149–1184 (2011)
9. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
10. Miyato, T., Maeda, S.I., Koyama, M., Ishii, S.: Virtual adversarial training: a regularization method for supervised and semi-supervised learning. IEEE Trans. Pattern Anal. Mach. Intell. **41**(8), 1979–1993 (2018)
11. Ruder, S.: An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098 (2017)
12. Sabokrou, M., Fayyaz, M., Fathy, M., Klette, R.: Deep-cascade: cascading 3D deep neural networks for fast anomaly detection and localization in crowded scenes. IEEE Trans. Image Process. **26**(4), 1992–2004 (2017)
13. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
14. Stekhoven, D.J., Bühlmann, P.: Missforest—non-parametric missing value imputation for mixed-type data. Bioinformatics **28**(1), 112–118 (2011)
15. Trapeznikov, K., Saligrama, V., Castañón, D.: Multi-stage classifier design. In: Asian Conference on Machine Learning, pp. 459–474 (2012)

16. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, pp. 1096–1103. ACM (2008)
17. Viola, P., Jones, M., et al.: Robust real-time object detection. Int. J. Comput. Vis. **4**(34–47), 4 (2001)
18. Yang, Y., Hospedales, T.: Deep multi-task representation learning: a tensor factorisation approach. arXiv preprint arXiv:1605.06391 (2016)
19. Yoon, J., Jordon, J., Van Der Schaar, M.: Gain: missing data imputation using generative adversarial nets. arXiv preprint arXiv:1806.02920 (2018)
20. Zhang, Y., Yang, Q.: A survey on multi-task learning. arXiv preprint arXiv:1707.08114 (2017) 3
21. Zhou, J., Liu, J., Narayan, V.A., Ye, J.: Modeling disease progression via fused sparse group Lasso. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1095–1103. ACM (2012)
22. Zhu, J., Chen, N., Xing, E.P.: Infinite latent SVM for classification and multi-task learning. In: Advances in Neural Information Processing Systems, pp. 1620–1628 (2011)