Explore the Deep Learning Models with Extrasensory Dataset

Zifan Xu
zfxu@utexas.edu
University of Texas at Austin
Austin, Texas

Yang Hu
yang.hu@utexas.edu
University of Texas at Austin
Austin, Texas

Xuan Wang
xxxuan0213@utexas.edu
University of Texas at Austin
Austin, Texas

Seungmin Kang
amernihi@utexas.edu
University of Texas at Austin
Austin, Texas

ABSTRACT

This paper illustrates the methods and result of exploring the ExtraSensory dataset. From this data set, our plan is to explore feature selection method among all provided features (225 features), and different methods including MLP(Multi Layer Perceptron), RNN(Recurrent Neural Network) and Classic Machine Learning method(Random Forest) to get better performance in classifying multi-label. In each section, the methods will be introduced, which is followed by the exploration process and result. At the end, we summarize the best way to predict in our knowledge and share the things we learned and things to pay attention to in the future.

KEYWORDS

Extrasensory dataset, neural networks, multi-label, missing labels

1 INTRODUCTION

Our work is based on Vaizman's work [7] published in 2017. Vaizman's team built the ExtraSensory dataset and shared it publicly in CSV file format. This data consists of sensory data and labels and it was collected from 60 participants for approximately 7 days. For each user, it has thousands of instances, typically taken in intervals of 1 minute. Every instance contains measurements from sensors from the user's personal smartphone, eg. accelerometer, magnetometer, location, and phone-state, as well as an accelerometer and compass from an additional smartwatch that the data collector provided. Each instance has at most 51 self-reported labels inputting the users, including body state labels such as (Lying down", "Sitting", "Standing"), Home-activities ("Cooking", "Cleaning", "Doing laundry") and so on. They utilized multiple layer perceptron (MLP) models to explore the best network structure and hyperparameters and get the best prediction in terms of the given test set.

Since the data set is incomplete and unbalanced labeling, they trained the model with multi-label instance weighting. And also used sensor-dropout to make the model resilient to missing sensors in addition to adding Frobenius norm on the loss function to prevent overfitting further. And also, since conventional accuracy is not suitable to evaluate unbalanced dataset, they used balanced accuracy(BA=(sensitivity+selectivity)/2). And they tried the combination across the depth of the MLP(one or two hidden layers) with dimension of hidden layers(width = 2,4,8,16,32). From this exploration, they got the best result, BA = 0.773 with MLP(16,16) model.

Based on this model, we tried to modify the construction of the hidden layers, the loss function, hyper-parameters, and additing regularization layers to further improve the model. As a result, some methods suit better for the circumstances and others do not. And we evaluate how each process takes place and show the performance of our model at the end.

2 FEATURE SELECTION

Extrasensory dataset has features extracted from ten types of sensors. Except for phone state sensors which have discrete categories such as battery state and WiFi availability and the categories have different sources of data, other sensors have many extracted features coming from the same source of time-sequential raw data. Taking accelerator as an example, it has 26 features extracted from its raw data and these features may be highly related because of the homology of the features (fig. 1). Such a high correlation turns out to be very common among all 225 feature pairs, which leads to the necessity of feature learning.

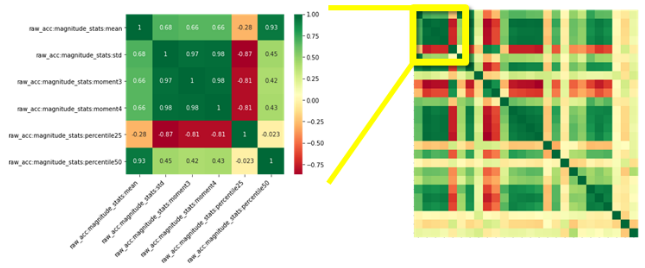


Figure 1: Correlation heatmap of 75 features from Acc sensors. The zoomed image is a detailed map of the first six features. The number in each entry represents the correlation coefficient.

2.1 Sequential Forward Selection

Algorithm: First, we try a classical wrapping method of feature selection (SFS): sequential forward selection. The wrapping method selects the features by the score of an evaluation algorithm. Here we use the multi-layer perceptron (MLP) model as the evaluation algorithm and use the balanced accuracy to score the performance of the model on the validation set. The hyper-parameters of the MLP: learning rate 0.00005, mini-batch size 300, hidden layer size [16, 16], number of epochs 20. We covered the algorithm in class and we used mlxtend package[6] to implement SFS.

Selection Result: We did such a selection twice to select all the 225 features, and validation scores versus the number of features selected is shown in fig. 3a.

The validation score has a significant increase when we select the first 50 features and almost saturates after we select 80 features. The score will not drop even though we add more features into the subset. The maximal score is around 0.825. We repeated the result using the selected features on the test set and compare it with the randomly selected the same number of features. All the random selections scores are averaged over ten times. The result is shown in fig. 3b. The selection result perfectly repeats the tendency of validation score and no obvious overfitting is observed. Compared with the random selection, the subsets selected by SFS are much better than the random selection with a maximal gap of around 0.05. With selecting more features, the gap becomes smaller.

Finally, we compared the result with the selection by sensors. In the related work done by Vaizman et al. [1], they select by core sensor as we mentioned above. We repeat the result using this strategy, which is shown as a red crossing in fig 2b. The red cross shows a test score of 0.823, which is similar to the best score achieved by SFS. But SFS can get such a score with only 80 features. Surprisingly, the selected features from two different trials are very different. It maybe because the features are interchangeable, and it will give similar result if we select any of a feature from a cluster of features that are highly related.

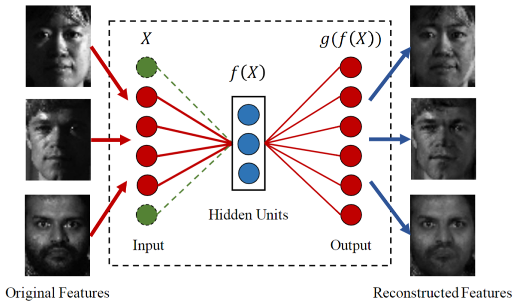


Figure 2: A diagram of auto-encoder network structure

2.2 Auto-encoder Feature Selection

SFS is a wrapper based method involving evaluation algorithms, which is quite time-consuming and requires labeled data. As we mentioned before, the 51 context labels involve large portions of unlabeled data and the classes in a label are very unbalanced. These all lead to a poor reliability of selecting the features by the evaluation algorithm. Unsupervised feature selection methods can be more essential in this case. Here we introduce a more efficient unsupervised feature selection method using an auto-encoder network for selecting features with high representability, which is similar to the network described in [2]. Fig. 2 shows a diagram of such an auto-encoder network.

Due to the limitation of time, we fixed the hyper-parameters of the network as follows. The network takes all the 225 features as input and outputs the same number of units with only one hidden layer with 20 hidden units. The activation function of the hidden layer is sigmoid function and a simple linear transform is applied for the output layer.

In the learning process, auto-encoder is simply described as minimizing a loss function as follow:

$$\Gamma(\Theta) = \frac{1}{2m}||X - \hat{X}||_F^2 + \frac{\beta}{2}\sum_{i=1}^{2}||W^{(i)}||_F^2$$

Where m is the number of samples and $||.||_F$ is the Frobenius norm for the matrices. $\hat{X}$ is the output of the network with exactly the same shape as the input. The second regularization term is β the regularization constant times the sum of Frobenius norms of the two weight matrices.

After training such a network, the features are selected by computing the magnitude of the weight related to each of the features. Denote the weight matrix connecting the input layer and the hidden layer $W^{(1)} = [w_1, ..., w_d]^T$, where the i-th row w_i corresponds to the i-th feature x_i. If $||w_i||_2 \approx 0$, the i-th feature contributes little to the representation of the other features; on the other hand, if the i-th features plays important role in the representation of other features, $||w_i||_2$ must be significant. So we utilize $||w_i||_2$ as the score to select the features that can represent other features.

We trained the network with all the training data and plotted the distribution of scores of features in fig. 4. The distribution shows that a plenty of features have scores close to zero indicating that they can be represented by other features. The algorithm selects features by scores from high to low ranks. Fig. 2c compares the evaluation score of auto-encoder selections using MLP model with SFS and random selections. Auto-encoder selections performs similarly to random selection and overwhelm a little bit when more than 80 features are selected.

Auto-encoder feature selector is much faster compared to the wrapped methods like SFS with an acceptable performance. So in general, auto-encoder feature selector is definitely an algorithm that deserves to try in this classification problem.

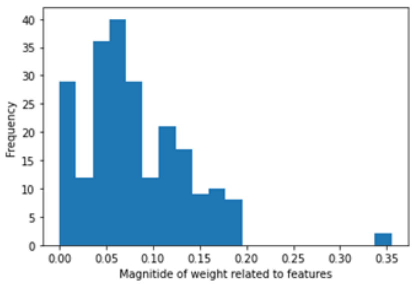
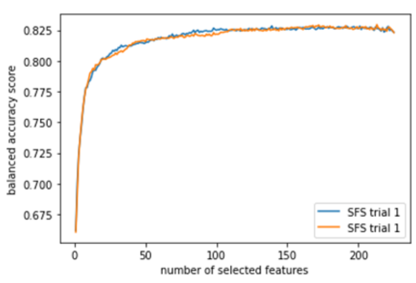


Figure 4: Distribution of features scores

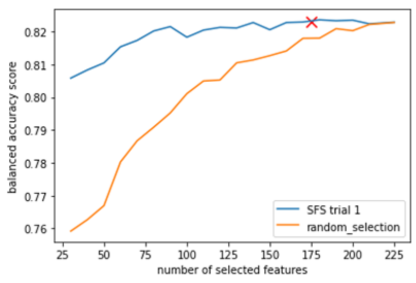
3 COMPARISON OF METHODS

3.1 Random Forest Model

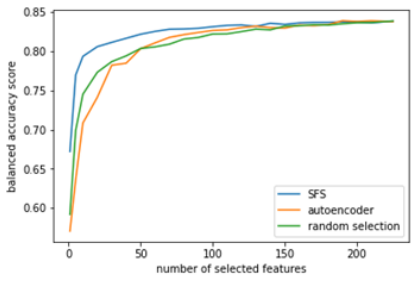
The random forest model is built to set a baseline and therefore compare the result with using neural networks.At the beginning of the modeling process, we only tried to output the six core labels, and it turns out a model with maximum depth of three gives a high



(a) Validation scores versus number of features



(b) Comparison between SFS and random selection on test set. Red crossing indicates the score of selection by sensors



(c) Comparison between auto-encoder feature selection and SFS and random selection.

Figure 3: Scores of feature selection by three different strategies: SFS, random selection and auto-encoder feature selection

accuracy to predict six labels. Then, the prediction extends to all 51 labels, which also includes more instances that have many missing labels. To deal with such cases, the sample weight is calculated according to the portion of missing labels. At the same time, to deal with the problem of unbalanced classes, we added additional parameters in the model to set it as 'balanced'. When evaluating the model, we used balanced accuracy as mentioned in the introduction and ignore the missing labels for the calculations.

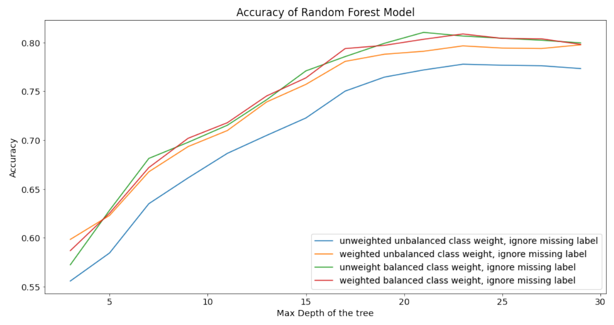


Figure 5: Random Forest result shows the effectiveness of sample weight (weighted) and balanced class (balanced).

To see if we really need to use sample weight and balanced class, and also what the depth of the final tree we want, we plot the "Max Depth vs. Balanced Accuracy" (Figure 5) with all different cases. As the figure shows, either including the balancing parameter to adding the weight will give us much improvement on the performance of the model. As the graph also shows, the random forest model performs the best when the maximum depth of the result tree is set to be either 21 or 22. When the depth increases furthermore, the performance on the test set of predicting is even worse due to over-fitting.

3.2 MLP model

As the most fundamental and convenient deep learning method, MLP model was first introduced to our multi-label classification problem.

Our MLP model processes an input feature vector $x \in \mathbb{R}$ with a sequence of J affine transforms. And we trained a multi task model, with multiple outputs, for a whole set of L binary labels. For hidden layers, we use a leaky rectified linear unit as activation function $g(v) = \max[\frac{v}{10}, v]$ for the output layer, we used the logistic function (sigmoid: $g(v) = \frac{1}{1+e^{-v}}$), to produce valid probability outputs. The actual binary predictions are achieved by thresholding the continuous outputs by 0.5.

We defined the function f as processing a batch of N examples, $f : \mathbb{R}^{N\times d \rightarrow [0,1]^{N\times L}}$. This function is parametrized by the free parameters of the model - the weight matrix and bias vector of each affine transform $\Theta = \{W_j, b_j\}_{j=1}^{J}$.

Training is done over a training set of N examples that have sensor features and incomplete labeling . We denoted the training set with feature matrix $X \in \mathbb{R}^{N\times d}$, the ground truth labels matrix $Y \in \{0,1\}^{N\times L}$, and the missing label matrix $M \in \{0,1\}^{N\times L}$. To train the model, we define the optimization problem in minimizing the loss function defined as follow:

$$Loss = \min_{\Theta}\left(\frac{1}{NL}\sum_{i=1}^{N}\sum_{l=1}^{L}\Psi_{i,l}c(f(X)_{i,l}, Y_{i,l})\right) + \lambda\phi(\Theta)$$

Where $c(\tilde{y}, y) = -(ylog(\tilde{y}) + (1-y)log(1-\tilde{y}))$ is the traditional cross entropy loss, λ is the regularization constant and $\phi(\Theta)$ is the Frobenius norm of the weight matrices of the network. Ψ is the instance weight coefficient, which is defined as $\Psi = W \times (1 - M)$. Where W is used to balance the two classes, $W_{i,l}$ is inverse proportional to number of instances with the same class as $Y_{i,l}$; M is the missing label matrix. For an entry (i, l) that are regarded as "missing label" ($M_{i,l} = 1$), $\Psi_{i,l}$ is set to zero to make sure this example-label pair contributes nothing to the total cost. By this weighting matrix, all the entries are normalized for each label, so that total contribution of positive examples is equal to the total contribution of the negative examples.

We used early fusion of the sensors. The training was done using gradient descent with back-propagation, for forty epochs, with mini-batch size of 300 examples. The learning rate was 0.0005.

In order to get the best result, we explore three different regularization methods. 1) drop out 2) batch normalization 3) Frobenius norm on loss function. We add variables to change drop out rate (0

Table 1: Hyper-parameter Searching Result

Dropout	Batchnorm	Alpha	BA
0.00	On	0.01	0.8375
0.00	On	0.00	0.8370
0.00	Off	0.00	0.8363
0.00	On	0.10	0.8363
0.00	Off	0.10	0.8362
0.00	Off	0.01	0.8360
0.15	On	0.01	0.8321
0.15	On	0.10	0.8317
0.15	On	0.00	0.8315
0.15	Off	0.01	0.8311
0.15	Off	0.10	0.8309
0.15	Off	0.00	0.8303

or 0.15), state of batch normalization (On or Off) and regularization parameter alpha(0 or 0.1 or 0.001) which is related to the Frobenius norm. The hyper-parameter searching is shown in table 1.

Among 12 combinations of hyper parameters, the combination with dropout layer shows relatively bad performance compared to the combination without dropout layer. While drop out layers seems not that helpful, Batch normalization seems helpful, so we did another iteration for batch normalization. We compared the effect of batch normalization while fixing drop out and alpha as 0, through various numbers of nodes (fig. 6). When there are [16,16] nodes, there is little difference, but when [256,256], there is a relatively big gap between 'On' and 'Off'. And we got best performance with [256,256] hidden layer as below. We interpreted the course of this result as that batch normalization speeds up learning by adjusting and scaling all input between each layer.

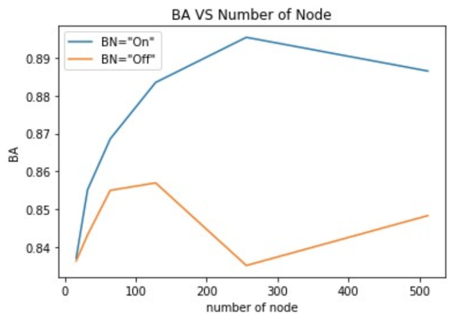


Figure 6: Compare batch normalization 'on' and 'off' in terms of different hidden layer size

In conclusion, among those three, batch normalization showed the biggest improvement by preventing overfitting considering that we got best Balanced accuracy = 0.895510, with batch normalization ='On', [256,256]nodes. However, the drop out method and Frobenius norm(regularization parameter) on loss function didn't show significant improvement compared to batch normalization.

3.3 RNN Model

In the previous paragraph we have introduced the MLP model, but the functions that MLP can achieve are still quite limited. For this Extrasensory data set, the labels have a complex temporal correlation with each other, and the length is various. This relationship is not concerned by MLP. The key point in RNN is that the hidden state of the current network will retain the previous input information and be used as the output of the current network. We built a simple RNN model to see if there any improvement of performance compared to other models. Similarly by using the instance-weighting matrix, we can ignore the effect of missing labels during training. To implement a basic model in comparison, we added a dense layer, which is the regular deeply connected neural network layer and the most common and frequently used layer, set the activation function as "sigmoid" together with a 0.2 dropout rate. From this model we got a balanced accuracy around 0.68 from the selected 175 features.

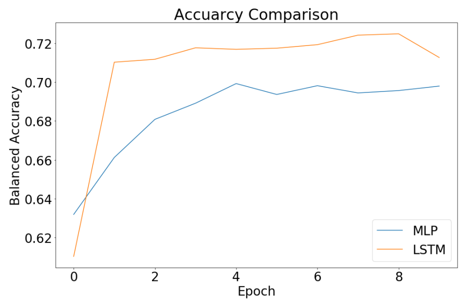


Figure 7: Comparing simple MLP model to RNN with LSTM model

In addition to the basic model, we extended it to RNN models with the Long Short Term Memory networks, known as "LSTMs". Since LSTM has the ability to avoid long-term dependency problems, it might provide a better memory structure for processing the 175 input features. We inserted a LSTM layer into our basic RNN model while keeping all the other hyper-parameters unchanged to see if it can help us to improve our model performance. As a result, the balanced accuracy is higher from the beginning and reaches to about 0.71 with the LSTM layer. We should pay more attention to the training part in the future work in order to get a more accurate and saturate BA.

4 CONCLUSIONS

This project is based on in-the-wild dataset, all the data was collected from users' regular natural behavior, which means the dataset has a large scale as well as a rich variability and flexibility. So feature selection occupied a large proportion in our project. After applying sequential forward selection and auto-encoder to the data, we found that balanced accuracy almost saturated when we selected around 80 features. However, selecting more features doesn't hurt the performance of MLP models.

Training deep neural networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. To solve this problem, we used different parameters and regularizations and found that batch normalization works the best, which also helps to reduce overfitting. We proposed our best MLP model with hyper-parameters of regularization parameter alpha = 0.0, epoch = 40 and learning rate = 0.0001 with two hidden layers that each has 256 hidden units and two batch normalization layers.

5 DISCUSSION

From this project, we have a better understanding of how to select features in different ways and get familiar with modeling using random forest (using RandomForestClassifier in sklearn [5]) and neural networks (using torch [4] and Keras [1]). During training the models, we learned how to deal with a dataset with many missing labels and extremely unbalanced class. These two problems confused us at the beginning because the accuracy looks extremely good with not much training. This is where we experience the case that was shown in class that high accuracy does not mean good models. Therefore, evaluation in this project used balanced accuracy (BA), which is a more fair replacement for the naive accuracy for unbalanced, sparse data. And BA shows much stable results in our experiments, compared to sensitivity and conventional accuracy alone. More broadly, BA is significantly useful when data has biased distribution.

Since we run all code on our own device and not using GPUs, the speed of processing feature selections and modeling was slow for a lot of the time. We adjust different parameters and try to make the processing time within a reasonable range. Nonetheless, programs such as the sequential forward feature selection still takes about 10 hours. However, this also pushed us to learn how to achieve our objective to about the same degree but spending less time waiting for the result. Thus, Auto-encoder is a way we learn to find features that help the model perform better and spending a lot less time.

We also learned a lot about neural networks when we model. Neural network was only a vocabulary we heard before we started, and now we know how to build neural networks with different shape of hidden layers, different activation functions including but not limited to PReLU, ReLU, Sigmoid, Tanh, LeakyReLu, and softmax, and different regularizations including regularization terms in loss function, batch normalization layers and dropout. We tested convolutional layers but did not work out well. Then we also tried recurrent layers and there are improvements to some extent but not concrete yet. We tried different optimizer and self-defined metrics.

Overall, we tried different techniques and used the ones that can improve the performance the most during validation. For the future work, the methods could be extended to semi-supervised learning, unsupervised learning and other architectures for neural networks such as ConvLSTM2D [1] and DeepConvLSTM [3] that could perform better for human behavior recognition techniques.

REFERENCES

[1] François Chollet et al. 2015. Keras. https://keras.io.

[2] K. Han, Y. Wang, C. Zhang, C. Li, and C. Xu. 2018. Autoencoder Inspired Unsupervised Feature Selection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2941–2945. https://doi.org/10.1109/ICASSP.2018.8462261

[3] Vishvak S. Murahari, Thomas Plötz, Vishvak S. Murahari Georgia Institute of Technology, Georgia Institute of Technology, Georgia Institute of TechnologyView Profile, Thomas Plötz Georgia Institute of Technology, Keio University, Singapore Management University, and Karlsruhe Institute of Technology. 2018. On attention models for human activity recognition. https://dl.acm.org/doi/10.1145/3267242.3267287

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[6] Sebastian Raschka. 2018. MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. *The Journal of Open Source Software* 3, 24 (April 2018). https://doi.org/10.21105/joss.00638

[7] Yonatan Vaizman, Nadir Weibel, and Gert Lanckriet. 2018. Context Recognition In-the-Wild: Unified Model for Multi-Modal Sensors and Multi-Label Classification. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4, Article 168 (Jan. 2018), 22 pages. https://doi.org/10.1145/3161192