Floor Surface Classification with Robot IMU Sensors Data

Group 14

Jiang, Yueyu
UC San Diego
A53277031
y5jiang@ucsd.edu

Zhou, Weiwei
UC San Diego
A53273469
w1zhou@ucsd.edu

Zhang, Song
UC San Diego
A53275617
sozhang@ucsd.edu

Gao, Shang
UC San Diego
A53275724
shgao@ucsd.edu

Abstract

In order to work intelligently and flexibly, robots should have the ability to navigate themselves. In this situation, understanding the surroundings becomes one of the most important tasks that helps robots to get reasonable reactions for navigation. Our project is aimed at establishing a floor surface classification model with the data collected by Robot Inertial Measurement Units (IMU). With several advanced feature engineering techniques, our customized Convolutional Neural Network beats a bunch of popular machine learning algorithms including KNN, Logistic Regression, Random Forest and LSTM with an multi-class accuracy of 80.15% *on the test dataset.*

1. Introduction

On different floor surfaces (carpet, tiles, concrete, etc.), robots can take different actions to move quickly and efficiently due to friction magnitude and bounce amplitude. If robots are able to figure out different types of the floor, it will be very helpful for the navigation of the robots.

The difficulties in the project can be concluded in three aspects. Firstly, the training dataset is relatively small (around 3k) so that the performance of deep learning model may not meet the expectation due to over-fitting or some other issues related to lack of data. Secondly, we have no prior ideas about how the robot is moving in general, whether moving straight like a robot car or balancing like a humanoid Robot, so that we can only explore the moving patterns based on the data without any hypothesis. Thirdly, the data is recorded in time series and noises exist, so we need to apply some denoising techniques for feature engineering.

Our goal is to evaluate both classic machine learning methods and deep learning methods, then making a comprehensive comparison between model candidates. With the help of some advanced engineered features on time series data and experiments on different models, we have got some satisfying results.

2. Related work

In this section, we will have a brief overview of the related work about the feature engineering and classification on time series data using different kinds of models and techniques.

2.1. Vibration Feature Extraction

The data in time series can be treated as a period of vibration. For classification on time series data, features extractions are performed in time domain, frequency domain and time-frequency domain. Among them, frequency related domain is both time consuming and resource consuming. René-Vinicio *et al*. [19] combined Chi-square, ReliefF and information gain feature ranking methods to select the optimal number of feature set for vibration in time domain. René-Vinicio *et al*. [22] presented the most relevant features of time domain for vibration using K-nearest Neighbor (KNN) and Chi-square method and tested accuracy of the classifier based on these features. In their research, the classification accuracy for vibration based on the top 10 selected features of time domain can reach approximately 95%.

2.2. Convolutional Neural Networks for time series data classification

Over the past few years, convolutional neural networks have achieved great performance in many areas, such as computer vision [15, 21, 20], natural language processing[14, 23, 11] etc. And CNN has shown its incomparable ability on classification. Inspired by this success, in 2014, Zheng *et al*. [25] applied deep convolutional neural networks on time series classification. In their work, they proposed a multi-channels convolutional neural networks, which is directly applied on raw data. Unlike previous algorithms, the proposed method does not rely on hand-crafted features. It can learn the classification objective function

directly from time series data. In the networks structure, convolutional layers are used to extract data features automatically and MLP is then applied on the hidden variable to perform classification. Specifically, they modified the traditional CNN by splitting data into univariate series units and learn data pattern from these units individually. Patterns are learnt by convolutional operations with bias and activation functions, which are Sigmoid and Tanh. This approach outperforms previous state-of-the-art algorithms on time series data classification at that time. Later, in 2016, Cui *et al*. [12] proposed an end-to-end networks model with similar structure as [25] but has multi-scale input for CNN. In the proposed method, input data is preprocessed to get multiple extra branches, which have different time downsampling rates. This approach take the high frequency perturbations and random noise into consideration and thus enable better performance on real-world applications.

2.3. LSTMs for time series data classification

Long Short Term Memory networks are a special kind of RNN. Typically they are used to do time series prediction. For example, Zheng *et al*. [24] used LSTM to do short-term traffic forecast. Actually, LSTMs can also be used to do classification tasks. Zachary *et al*. [17] used LSTMs to recognize patterns in multivariate time series of clinical measurements by replicating targets at each sequence step, and do multilabel classification. Their models outperformed several strong baselines, including a multilayer perceptron trained on hand-engineered features. Also, some people tried a combination of LSTM and CNN to deal with classification tasks. Soo *et al*. [9] used LSTM layers extract the sequential information from consecutive audio features, and CNN layers learn the spectro-temporal locality from spectrogram images. Then, they combined them together and make predictions.

3. Dataset and Features

In this section, we talk about data analysis and various feature engineering aspects.

3.1. Dataset Analysis

The dataset is a public accessible dataset on Kaggle[1], provided by Heikki Huttunen and Francesco Lomio from the Department of Signal Processing and Damoon Mohamadi, Kaan Celikbilek, Pedram Ghazi and Reza Ghabcheloo from the Department of Automation and Mechanical Engineering both from Tampere University, Finland. They collected the sensor data from the Inertial Measurement Units (IMU sensors) on robots in time series, shown in Figure 1. An IMU sensor is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using a combination of accelerometers and gyroscopes, sometimes also

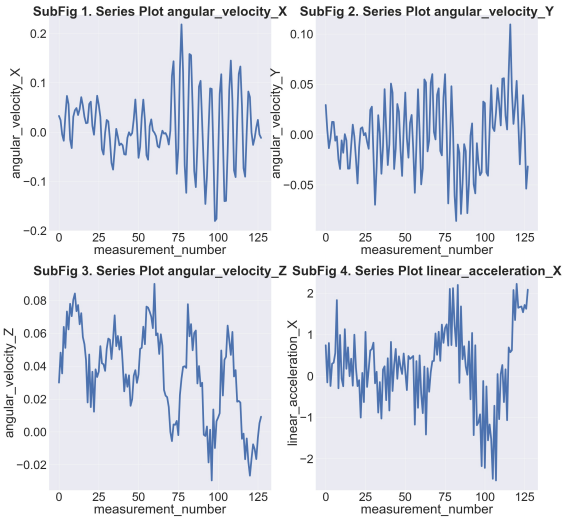

Figure 1. Visualization of Data Examples

data	train	test
number of measurements	487680	488448
number of time series	3810	3816

Table 1. size of train data and test data

magnetometers[2]. The surface types vary in different time series.

For the size of dataset, shown in Table 1, there are 3810 time series and 128 measurements in each time series in train data. And there are 3816 time series and 128 measurements in each time series in test data. Each measurement is presented as one row in X_train.csv and X_test.csv. The surface types of each time series are recorded in y_train.csv. We can match and merge the measurements of each time series and the corresponding surface type in train data using series_id as foreign key.

For features, each measurement records the data from 10 sensor channels: 4 orientation channels, 3 angular velocity channels and 3 acceleration channels. The orientation channels encode the current angles how the robot is oriented as a quaternion. Angular velocity describes the angle and speed of motion, and linear acceleration components describe how the speed is changing at different times. For classification targets, there are 9 different surface types in train data, the distribution of which is shown in Figure 2.

3.2. Feature Engineering

Before doing feature engineering, we do some data preprocessing.

Firstly, we check for NULL data. Fortunately, there is

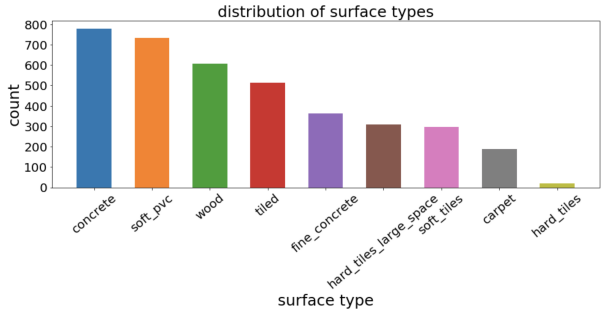

Figure 2. Distribution of Surface Types in train data

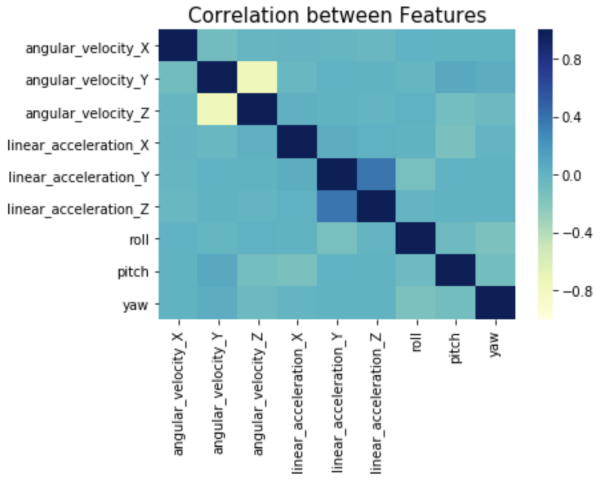

Figure 3. Correlation between Features

no NULL data in the dataset. Secondly, we transform 4 orientation features from quaternions to Euler angles[3], $'roll'$,$'pitch'$ and $'yaw'$. Although quaternions are more rigorous in mathematics and provide an alternative measurement technique that does not suffer from gimbal lock, they are less intuitive than Euler angles for thinking about actual cases and the math can be a little more complicated. So transforming quaternions to Euler angle is more helpful.

Figure 3 shows the feature correlations of training data after changing angle representation. We can notice that there is a strong correlation between $'angular\_velocity\_Z'$ and $'angular\_velocity\_Y'$. And the feature correlations of training data and test data are almost the same.

Then we do feature engineering and extract useful features for each time series.

3.2.1 Statistical Feature

As we take a deep research into the dataset, we have find that feature value follows different distributions if they belong to different classes. As the figure 1 shows, some of them belong to normal distribution, some of them belong to multimodal distribution. These distributions have different means, variations, modes, quantiles, maximums and so on. Given that we are classify a time series which means we would classify a set of observations instead of a single. A naive strategy is to extract the statistics in these time series, then predict this series to the class with the most similar distributions. Follow by this strategy, we designed a bunch of statistics feature, including the classic $mean$, $median$, max, min, $range$, std(standard deviation), $absolute\ max$, $absolute\ min$. And for more accurately curve the distribution, we added more customized features including: mean, maximum, minimum of absolute values, maximum divided by minimum, mean of the changes between neighbors. These features provide similarity metrics between distributions.

3.2.2 Fourier Analysis

As we exploded the data, we find the features will follow different developing trending in each series. And their developments look very like some kind of wave. To extract their developing information, we did fourier analysis in these series. Fourier analysis consider each series as an addition of waves with different frequencies. Fourier analysis give us a clear view to the developing trend in each series, as the figure shows, most of the amplitude concentrate on some specific frequencies. For each features in each series, we take their amplitudes in the main frequencies as their FFT (fast fourier transform) feature.

3.2.3 Vibration Features

Since the features from sensor channels are recorded in many time series, we can treat the continuous measurements of each feature in a single time series as an independent vibration. So we can also extract some useful vibration features from different vibrations. Firstly, we explore about the vibration patterns of different surface type and notice that they vary a lot from each other. Secondly, based on the result mentioned by René-Vinicio et al. (2018)[22], we extract the top 10 relevant features of time domain for vibration on each channels. The top 10 features are $kurtosis$, $mean$, $CPT5$, $skewness$, $slope\ sign\ change$, $wave\ length$, $norm\ entropy$, $square\ root\ amplitude\ value$, $mean\ of\ absolute$ and $zero\ crossing$.

4. methods

In this section, the classification methods and network architectures are stated in detail.

4.1. k-Nearest Neighbor (KNN)

As we mentioned in Section 3.2.1, a basic strategy for this problem is to make prediction based on similarity be-

tween feature distributions. K-Nearest Neighbor is such an algorithm to extract the similarity between samples. For each sample in testing set, k-Nearest Neighbor find the k samples in training set has the most similar feature value, denoted as its neighbors. Then predict with the label appear most frequently in the neighbors using Euclidean distance[13] (Equation 1).

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{1}$$

KNN is a simple model, but it matches the observation we have in data exploration, so we put it as a baseline model.

4.2. Logistic Regression (LR)

Logistic regression is a widely used algorithm to solve classification problem. It used maximum likelihood to estimate the parameters of logistic model, a kind of generalized linear model. A logistic model uses a logistic function (Equation 2) to map the output of a linear model, so it can generate a binary output which is suitable for classification problem.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

Using logistic regression helps to model a probability of an event. But usually it only takes binary or bipolar features, to suits for logistic regression, we discretized our continuous features. For each continuous feature, we collected their value range in the training set, split this range into multiple (typically 50) bins, then create binary features with the same number to indicator whether the statistic feature value of a series fall into the bins.

4.3. Random Forest (RF)

Random forest is a commonly used classification methods. Random forests [10], introduced by Breiman, are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Predictions are made by averaging the predictions from all the individual regression trees, which can be concluded in Equation 3.

$$f(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x) \tag{3}$$

By comparison to single decision tree, using random forest can efficiently avoid over-fitting. What's more, RF is relatively robust to outliers and noise, which is beneficial for the vibration data. Also, from the result of RF classifier, we can obtain some internal estimates of error, strength, correlation and variable importance, which can provide a deep understanding of the extracted features.

4.4. Convolutional Neural Network (CNN)

One dimensional convolution is to perform the following equation

$$(s * k)[n] = \sum_{m=0}^{l} s[m]k[n-m] \tag{4}$$

where s is the data we want to apply convotion operation on, k is the convolution kernel, l is the length of the signal.

The convolution operation can be shown as Figure 4, which is also the structure of one convolutional layer in our network. Consider the convolutional kernel as a filter, by modifying the value of the filter, it can be activated by different patterns. In our network, we apply convolution kernels with multiple channels so that one layer of convolution can extract several low level features from the input. These simple features can then be used to form more complex patterns within higher layers. So with hierarchy structures, we are able to get abstract and complex features.

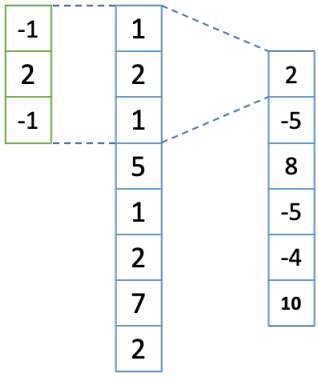

Figure 4. One dimension convolution

In our work, we consider time as the dimension we apply the convolution operation on. We assume that different types of floor will force the robots to perform differently, which would be represented as different change patterns of the data through time. And as we mentioned above, the data on time domain and frequency domain carry meaningful information of the floor types, so we will exploit both of these information. Our approach will be detailed in the following paragraphs.

The overall architecture of our network has three sequential stages: transformation stage, convolution stage and MLP stage.

1) The transformation stage performs some data preprocessing.

2) The convolution stage applies two separated convolutional neural networks on the two branches of the data and learn the features from the two branches individually.

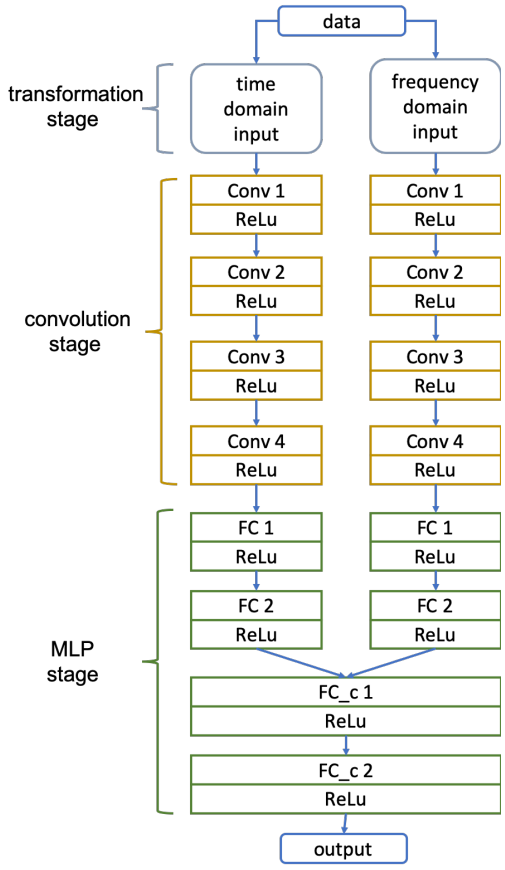

Figure 5. Architecture of CNN (the configuration of each layer is shown in table 2)

3) The MLP stage applies fully connected layers to perform classification. Separated fully connected layers are first used on the two branches respectively and then the output of two branches are concatenated together as the input of the following MLP.

The architecture of our networks is presented as Figure 5. Notice that we use dropout at each convolutional layers and Fully connected layers but it is omitted in the figure for simplification.

The activation function we used is ReLu [18], which is applied after each linear layer. The configuration of the network is shown as Table 2.

4.5. Long Short-Term Memory (LSTM)

Long Short Term Memory networks are a special kind of RNN. RNNs, also callled Recurrent Neural Networks, are networks with loops in them. This is where it differs from CNN.

Cells in LSTM networks have well-defined structures which enable them to memorize information. Memory cells in LSTM can bridge very long lags [16]. For each cell in an

Layer	Time Domain		Frequency Domain		Output Channels
	kernel size	stride	kernel size	stride	
Conv 1	8	2	8	2	32
Conv 2	9	4	7	2	64
Conv3	8	2	8	2	128
Conv4	4	1	3	1	256
	output channels				
FC 1	64				
FC 2	64				
FC_c 1	64				
FC_c 2	9				

Table 2. Configuration of CNN

LSTM network, there are several computations to do. For instance, suppose an LSTM cell takes C_{t-1} and h_{t-1} from the previous cell, and also takes xt as input. Then, first, forget gate would be calculated as:

$$f_t = \sigma(W_f \cdot [h_{x-1}, x_t] + b_f) \tag{5}$$

Then calculate how much information to store:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{6}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{7}$$

Now we can get calculate the output and pass it to the next cell:

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{8}$$

$$o_t = \sigma(W_o \cdot [h_{x-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_t \times tanh(C_t) \tag{10}$$

In our work, for each ID number for the measurement series, we have 128 measurements, each has 10 columns corresponding to the sensor. Our basic idea is that, robots on different kinds of floor will perform differently in their time-series behaviors. Therefore, even though LSTMs are typically used to do time-series prediction such as speeches and movies, perhaps we can try to use LSTM model to help us extract some time-series feature, and use them to do prediction.

The basic network architecture can be shown as the Figure 6. So, the LSTM layer takes input size 10, which corresponds to the input data. The hidden dimension is 256, so for each batch, the output tensor will be of size $batch\_size \times 128 \times 256$. Here we make $batch\_size$ to be 64. Then we take the statistical features from it, we can take mean, standard division, max value and min value, and

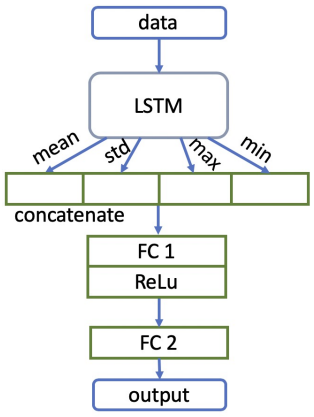

Figure 6. Architecture of LSTM model

concatenate together. Finally, pass this tensor to some fully connected layers and get the output.

5. Experimental Evaluation and Results

We use multi-class accuracy as the metric of our algorithms. Detail of multi-class accuracy is showed in equation 9, where N is the number of samples, p_i is the prediction to sample i, y_i is the label of sample i.

$$Acc = \frac{1}{N}\sum_{i=1}^{N} I(p_i == y_i) \tag{11}$$

5.1. Classic Machine Learning Models

We use k-Fold techniques[4] for cross-validation on classic machine learning models. We randomly shuffled and created 20 data folds.

For k nearest neighbors[5], after a series of tuning, we set the only hyper-parameter k the number of neighbors as 50. With $k = 50$, KNN achieved 49.26% multi-class accuracy in test dataset.

For logistic regression[6], we have 2 hyper-parameters to tune, the number of bins (noted as k) we created for discretization as we have mentioned in Section 4.2, and the l2 penalty parameter (noted as c). L2 penalty could help to improve over-fitting of model. With $k = 100$ and $c = 1.0$, logistic regression achieve 60.25% multi-class accuracy in test dataset.

For random forest[7], we have 2 hyper-parameters to tune, the maximum depth of the tree (noted as max_depth) and the number of features to consider when looking for the best split (noted as $max\_features$). With $max\_depth = 70$ and $max\_feature = 20$, logistic regression achieve 71.81% multi-class accuracy in test dataset. The top 5 features with the highest importance are $'yaw\_absolute\_max'$, $'yaw\_absolute\_mean'$, $'yaw\_norm\_enropy'$, $'yaw\_mean\_absolute'$ and $'yaw\_square\_root\_amplitude\_value'$. Obviously, features related to $'yaw'$ orientation value more importantly than other features.

The detail of our experiment results are showed in Table 3.

5.2. Convolutional Neural Network (CNN)

Firstly, we randomly split the training data into 2 parts: 80% to train the model and 20% for validation and hyper-parameters tuning in CNN experiments.

The original input data has 10 channels. We then convert the orientation channels data to Euler Angle, which reduces the input to 9 channels. Each channel is a sequence of length 128. So, the shape of input for the time domain branch is 9×128. We applied Fourier transformation on the 3 angular velocity channels and 3 acceleration channels, which produces the input of frequency domain branch with shape 6×65. To avoid overfitting problem, we apply dropout on each linear layer with ratio 0.5.

We use cross-entropy loss for this multi-class classification task. The loss is computed as

$$Loss = \sum_{i} y_i' \log(y_i) \tag{12}$$

where i denotes the class, y_i' is the true probability for the sample belonging to class i, y_i is the predicted probability for the sample belonging to class i.

The proposed CNN architecture is implemented on the PyTorch[8] deep learning framework, and the stochastic descent algorithm Adam is applied for an end-to-end training. The learning rate is set to 10^{-4} and we apply gradient clip with the max norm of the gradient set to 1. We train the network for 2000 epochs with a batch size of 10.

The highest accuracy on validating data is 84.38% and the highest accuracy on training data is 96.61%. As Figure 7. shows, accuracy increase as the training process going on and overfitting problem exists.

5.3. Long Short-Term Memory (LSTM)

The LSTM architecture is also implemented on the PyTorch[8] deep learning framework. In the experiment, we also use cross-entropy loss for this multi-class classification task. The batch_size here is 64, dropout is 0.2, learning rate is 0.001, optimizer is Adam. Also notice that early stop strategy is applied in the training process.

The training and validating accuracy along with training process can be shown in Figure 8. The highest accuracy on training data is 95.74%. However, when we test it on the testing data, it only get an accuracy about 52%. It suffers from over-fitting problem. One possible reason might be, LSTM may not be a very good model to extract features and do prediction in this task. Typically LSTM can be used

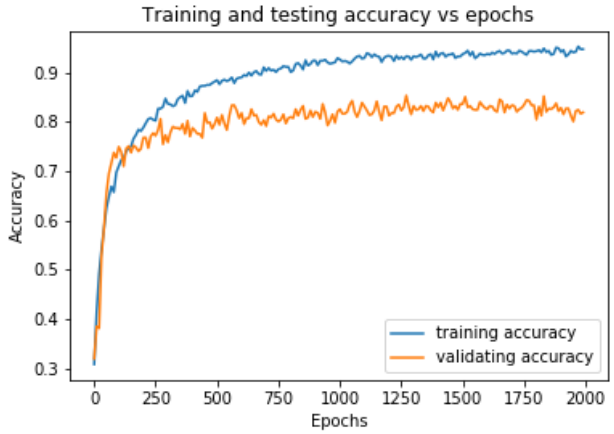

Figure 7. Training and Validating Accuracy vs Epochs of CNN model

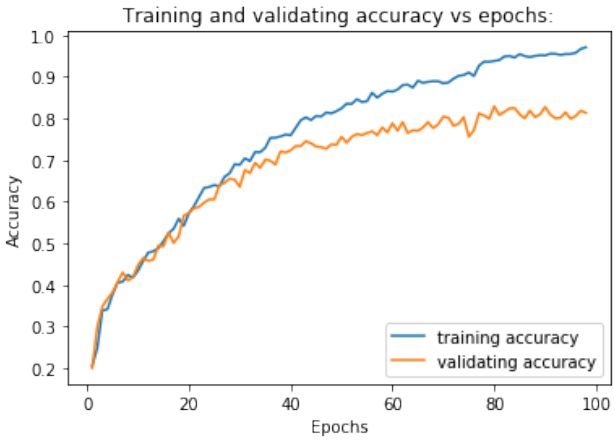

Figure 8. Training and Validating Accuracy vs Epochs of LSTM model

Model	Accuracy (training data)	Accuracy (test data)
KNN	49.29%	49.26%
LSTM	95.74%	51.82%
LR	100.00%	60.25%
RF	89.16%	71.81%
CNN	96.61%	80.15%

Table 3. Summary of Multiclass Accuracy on each Model

to take time-series information as input, and predict a value in the next moment. Probably it is not a very good model for this problem.

5.4. Summary

The summary of the multiclass accuracy on each model discussed above is listed in Table 3.

By comparison, CNN model outperforms the other models. The accuracy can reach 80.15% on the private test dataset of Kaggle, ranking No.15 over nearly 1500 participants.

We also examine the confusion matrix on training data of CNN model, shown in Figure 9. Obviously, dark colors are concentrated on the diagonal. The main confusion is between $'tiled'$ floor and $'concrete'$ floor. In common sense, both $'tiled'$ floor and $'concrete'$ floor are of relatively higher hardness and smoothness than other surface types, like carpet, wood, etc. Correspondingly, the moving pattern on these two floors will be similar, especially on vertical dimension. So, this confusion matrix is in line with reality to some extent.

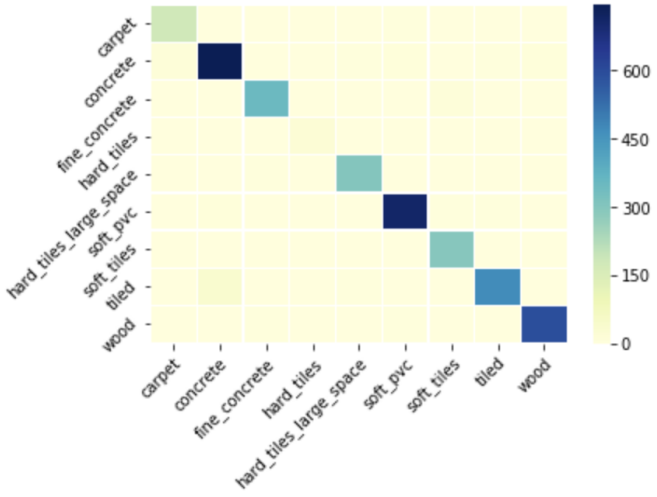

Figure 9. Confusion Matrix by CNN model

6. Conclusion and Future Works

In this project, we have tried 3 classic algorithms: KNN, logistic regression, random forest. KNN is a baseline model, it used each series neighbors to make prediction. Each prediction is made based only a small part of training set, it is highly vulnerable to the noise in the dataset.

Compared with KNN, logistic regression generate a maximum likelihood estimation to the series belonging to each classes, this utilization of totally training set could be the reason it beats KNN. While compared with logistic regression, random forest overcome the restriction that input must be either binary or probability. In logistic regression, sub-range indicator are created manually, however in random forest sub-range are created by maximizing information gain. This could be the reason random forest beats logistic regression.

Compared with 3 classic algorithms, CNN and LSTM provide an approach to automatically exploit wave feature

in the series and create more complex model. However, LSTM, a variant of RNN, is created to predict future initially, maybe this is the reason it didnt achieve a good result.

Given our model still suffer from insufficiency of data(most of them shows over-fitting). In the meantime there exists huge imbalance between samples number in different classes. Over-sampling techniques like SMOTE could be a promising direction to try. In addition, we believe deep learning model have automatically extracted some valuable features, applying these features on classic model like logistic regression and SVM may be an interesting attempt. Beyond improving models' performance, model interpretation is would be valuable. Visualization for classic model like logistic regression and random forest could help us understand how did our models make these prediction.

7. Contribution

For the main part of our project, the contributions of each author are listed as follows.

Song Zhang and Weiwei Zhou are responsible for the research on feature engineering and implementation of classic machine learning models (refer to Section 3, Section 4.1, 4.2 and 4.3, Section 5.1).

Yueyu Jiang is responsible for the design and implementation of CNN model (refer to Section 4.4, Section 5.2).

Shang Gao is responsible for the design and implementation of LSTM model (refer to Section 4.5, Section 5.3).

References

[1] https://www.kaggle.com/c/career-con-2019/.

[2] https://en.wikipedia.org/wiki/Inertial_measurement_unit.

[3] https://en.wikipedia.org/wiki/Euler_angles.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[8] https://pytorch.org.

[9] S. H. Bae, I. Choi, and N. S. Kim. Acoustic scene classification using parallel combination of lstm and cnn. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2016 Workshop (DCASE2016)*, pages 11–15, 2016.

[10] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[11] D. Chen, J. Bolton, and C. D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*, 2016.

[12] Z. Cui, W. Chen, and Y. Chen. Multi-scale convolutional neural networks for time series classification. *arXiv preprint arXiv:1603.06995*, 2016.

[13] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2005.

[14] D. Gordeev. Detecting state of aggression in sentences using cnn. In *International Conference on Speech and Computer*, pages 240–245. Springer, 2016.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzel. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[18] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[19] R.-V. Sánchez, P. Lucero, J.-C. Macancela, M. Cerrada, R. E. Vásquez, and F. Pacheco. Multi-fault diagnosis of rotating machinery by using feature ranking methods and svm-based classifiers. In *2017 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC)*, pages 105–110. IEEE, 2017.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[22] R. Snchez, P. Lucero, R. Vasquez, M. Cerrada, and D. Cabrera. A comparative feature analysis for gear pitting level classification by using acoustic emission, vibration and current signals. *IFAC-PapersOnLine*, 51:346–352, 01 2018.

[23] Y. Zhang, S. Roller, and B. Wallace. Mgnc-cnn: A simple approach to exploiting multiple word embeddings for sentence classification. *arXiv preprint arXiv:1603.00968*, 2016.

[24] Z. Zhao, W. Chen, X. Wu, P. C. Chen, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

[25] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.