ML Presentation

Hi everyone:
Our project is a Kaggle competition. I will introduce our project in four parts.

The first is introduction.

This competition is a **Multi-class prediction problem**, we should predict the adoption speed for specific pet from test set. The adoption speed has five different levels.

What is the data?

We have three types of data about the pets. The **main dataset**, **image and image metadata**, **description and sentiment data**. The **main dataset** is in CSV form. Each column is a feature about the pets. Of course there are some usefulness features such as RecscuerID and Pet Name. We just drop them. The **description attribute** is Profile for pets which contains several sentences mainly written by english. We put it into other file and will analyze it individually. **The image data** contains more than 50,000 pictures of these pets, some have multiple pictures while some have no pictures. The **image metadata** is the output of the these images analyzed by google's Vision API. As we can see in this JSON file. There are some Properties Annotated by Google's Vision API such as dominant colors. **The description data** is text data from the description attribute of main dataset I have mentioned previously. **The sentiment data** is the output of description data analyized by google's NLP API. For every sentence of the description, there are a sentiment score and magnitude. It also contains the document sentiment score and magnitude.

Ok, I will start to introduce the most exciting part. The impletation of our model.

This is the **overview** of our model's architecture. It is gorgeous, right? We have several different types of dataset now. Intuitively we can use two methods to achieve our goal. **Firstly**, we can extract the features from different types of datasets and then fusion them all. We could train a single model using these features and then predict the adoption speed. **The second method** is we can train different models for different types of dataset and then ensemble these models. What we do? We use these two methods both.
In general, we have trained four models. For the first model we use a **fine-tuned XGBoost**. The second model we use a **pre-trained VGG16**, then train our own last three Dense layers. This is what we call **transfer learning**. The three and four models are our **sequence models** to deal with text data. We will conbine the outputs from these models to get the final adoption speed.

Let's see our first model.

The input is the main CSV dataset and sentiment JSON data. For the CSV dataset, we do some feature selection firstly, then standard the numerical features and dummy the categorical features. For the sentiment data we manually construct **five features**. The max, minimum and mean score of sentences maltiply by maginitude. We also extract the document score and magnitude. Do not ask me why we manually construct these five features. It

is about intuition. A kind of important skill you don not have. **Then we fusion the these two types of features together and make it as the input of XGBoost**. Actually we have tried to trained a neural network to deal with this input. But the result is not really good as the XGBoost. We realize the neural network is not necessarily better than the traditional machine learning models, **especially in the case of insufficient data**, such as our current input, almost 10,000 data, It may also be not enough for the neural network. The traditional tree models have an irreplaceable role for some sizes of data.

The second model is our CNN model.

The dataset is the jpg images. For the stage of **data preprocessing**, we **augment** our image data through a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better. We don't have a powerful machine to load all the image data into memory so we implement the **real time generator** to generate our batch-size input sequentially. We have not enough image data and CPU or GPU computation time to trained our own VGG16. This is why we use transfer learning. We just train the last two dense layer and Softmax layer. For the **previous thirteen** convolutional layers and pooling layers, we use the parameters from ImageNet. Because the ImageNet dataset contains several "cat" classes and many "dog" classes among its total of 1000 classes, this model will already have learned features that are relevant to our classification problem.

The last model is our sequence model.

The dataset is text data of pets profile. The intuition is we can use some kind of Recurrent Neural Network to deal with this types of data. The first problem is how we can representing these words. The simplest way is to use the **one-hot encoding**. But it is not a good way because any two of these words will have the same similarity. In order for the representation of the word to contain **semantics**. we trained a Word2Vec model. The output of Word2Vec is the **embedding matrix** and we use it as the embedding layer of our sequence model. It is the first hidden layer. The second the third hidden layer we use **LSTM**. LSTM is always the second best choice for sequence data, of course, in my opinion. We use dropout to avoid overfitting. The last layer is Softmax layer to predict the five adoption speed.

The last question is how to combine the output of our models.

We simply use **voting results** of these three models. For samples without multiple outputs, we use the output of the model with the **highest accuracy**. For samples with three different voting results, we also use the output of the model with the highest accuracy.

The final part is the result.

The competition use the kappa score as the metric. These are our current kappa score. As can be seen in the picture, different model has different result and the ensemble result has the max score which is nearly 0.48. The best result from the Kaggle leadboard is 0.509. It can be concluded that our work is doing well and we will improve in the future.