Thalmic Machine Learning Challenge

Gesture Data Set

Stefan Schneider

23/11/2016

Hello Idris,

Please find below a summary of the process I took to create, save, and report the optimal machine learning classifier for the gesture data set. The document begins with a table summarizing each of models trained, followed by a description of the pipeline program. After this is a short description of each of the classifiers used along with their result. Lastly I describe the final steps of creating an ensemble model and the possible steps I would take to achieve further accuracy. I am confident from reading this document you will see my strengths as a data scientist. I look forward to hearing your feedback.

Stefan

Within the body of the text below A: refers to accuracy and F1: to the F1 score. 1.0 would represent a perfect accuracy or F1 score.

All code for this challenge was written using Python 3.5.

PredictGesture.py is the file used for classifying provided datasets.
In case you do not have TensorFlow I have included a PredictGestureNoTF.py file as well.

Please view the provided README.txt for additional details

Summary Statistics

Model	Accuracy	F1 Score
K-Nearest Neighbour	0.803	0.799
Random Forest	0.680	0.682
Gaussian Naive Bayes	0.674	0.662
Multinomial Naive Bayes	0.750	0.736
Logistic Regression	0.815	0.728
Linear Support Vector Machine	0.542	0.497
Radial Basis Function Support Vector Machine	0.164	0.067
Simple Neural Network	0.782	0.776
Deep Neural Network	**0.838**	**0.838**
LSTM Recurrent Neural Network	0.785	0.761
BiDirectional Recurrent Neural Network	0.721	0.716
Ensemble	0.825	0.823

Data Set Creation

The first data set I created took each example and created a 400 length feature vector by chronologically inserting the 8 response variables of the 50 time steps one after the other. The correct classification (gesture number) was then inserted at the end of the feature vector. The numpy.vstack function created the final dataset by iteratively adding each example feature vector as a row for all examples and gestures. During this process I checked for the length of each feature vectors where I found 14 examples that did not contain 50 time series entries. Due to a lack of context they were dropped from the final data set (Table 1). Example 434 and 1708 contained erroroneous data for numerous gestures. If this were based on real data I would consider investigating the unit that recorded these inputs. The final constructed data set had a size of 401 x 11986 and all the listed data above are reported using this data set.

After performing all the models listed above on this dataset I created a dataset using a polynomial array to try and further improve accuracy. For each example, the 8 feature values for each time step were converted into a polynomial array and then inserted as above. This created a 2251 x 11986 data set and was used for additional testing.

Initialize the Pipeline

Before training any classification models I wrote a general pipeline where the preprocessing and quality metrics would be universally applicable across models. When the program initializes the data are separated into features (X) and classifications (y). Utilizing sklearns train_test_split function the data was randomly separated into a training set, cross validation set, and test set (60%, 20% and 20% respectively). The user is then asked which model they would like to run: KNN, RF, NB, MNB, SVM, LOGIT, SNN, DNN, RNN, or BiNN corresponding to the names listed in the table above followed by any relevant conditional parameters (Ex. C parameter for SVM to adjust regularization, or for the Neural Networks epochs, regularization, and hidden layer size). Ignoring the models themselves, upon training completion the model is used to predict the classifications of the cross validation set. The user is then presented with quality metrics of accuracy (using the sklearn score function), weighted F1 score (using the sklearn metrics.F1_score function) and a confusion matrix (using the sklearn metrics.confusion_matrix function). A report is presented comparing the recent result to the results previously trained (if they exist). The user is then presented with an option to save the current model. If yes is selected, a pickle of the classification variable along with the reported accuracy, F1 score, and conditional parameters are saved.

Using the above information, each time the program initializes a summary statistics table is presented to the user containing the models, accuracies, F1 scores and conditional parameters. The user can then easily read and understand which models are saved, which are performing well, and how they may change the parameters of a more complicated model to improve performance.

The 20% testing set was used to test the ensemble model classification results.

The Learning Models

I started simple with K-Nearest Neighbour and Random Forest which are used for their robustness and computational efficiency. These models provide a good baseline understanding for how well your data may be represented using machine learning algorithms. K-Nearest Neighbour creates classifications by grouping models that are near each other within a high-dimensional feature space. At very high dimensions (>100-1000) this can cause problems as no data point is near another (known as the curse of dimensionality). Principle Component Analyses is one approach to dealing with this problem. For the gesture dataset, KNN was performed with sklearn's nearest class, and performed well without any modification (A: 0.803, F1: 0.799). A Random Forest was then performed using Sklearn's tree class. The Random Forest algorithm separates the given feature set randomly into branches utilizing feature bagging (random sampling with replacement). If a feature is a strong predictors within a branch it is utilized in the next branch of the tree. While useful for some data, Random Forests have a tendancy to overfit training data as a result of this. The final classification is determined by taking the mode of the predicted classification for each branch. The Random Forest overfit the gesture data (100% accuracy on the training set) and was a poor predictor of the testing set (A: 0.674, F1: 0.662).

Next I tried a Bayes approach. Naive Bayes models are also computationally efficient. Naive Bayes models use maximum likelihood and a Markovian approach which considers each feature as conditionally independent from each other features. The gesture dataset is obviously not independent, given that the data are a chronological time series however, due to their use of likelihood, Naive Bayes models have had success predicting complex and interdependent classifications. Gaussian Naive Bayes assumes feature variability follows a Gaussian distribution, while Multinomial Naive Bayes considers only that features are multinomial. Both models were performed using sklearn's naive_bayes package. The Gaussian Naive Bayes model performed similar to the Random Forest (A: 0.674, F1: 0.662). The Multinomial Naive Bayes model had a better accuracy and F1 score (A: 0.748, F1: 0.740).

After the Naive Bayes I considered a one-vs-rest Logistic regression. Logistic regressions utilize a sigmoid function, a logit function, and linear regression to categorize classifications between binary results. To deal with multiclassification data, the one-vs-rest approach essentially performs an analysis for each classification considering all alternative classifications as the opposing binary value. Logisitc regression was performed using the sklearn LogisticRegression class and was the best non-TensorFlow models when classifying the gesture data set (A: 0.811, F1: 0.736).

Increasing in complexity I then tried two support vector classifiers: Linear and Radial Basis Function (RBF). Support Vectors are known as large-margin classifiers and use a hinge-loss cost function rather than a sigmoid function to classify data. Support Vectors use vector inner products (also known as scalar product or dot product) to convert high-dimensional features into scalar values which are then used to optimize the distance away from a linear hyperplane. RBF SVMs are similar to Linear SVMs except instead of the dot product they use the a kernel function (also known as Integral Transform function) which allows for a nonlinear classification boundary. Support Vectors can also include a regularization parameter C which can adjust the strength of features for a given data set. Using the Sklearn svm class I trained both models on the gesture dataset. I personally have had poor results with SVMs. I find them computationally expensive, difficult to optimize using C, and rarely do they outperform simple algorithms such as K-Nearest Neighbour and Random Forest. A similar outcome occurred here where a linear SVM proved to contain a poor decision boundary (A: 0.542, F1: 0.497). The RBF SVM was unable to determine a pattern and reported all results a single classification despite alternative C values (A: 0.164, F1: 0.067).

After the above approaches I began training the data on neural network algorithms. Neural networks are powerful statistical tools able to capture relationships impossible to other models (ie. XOR relationships). Neural network are composed of an input layer (features), hidden layers, and output layer (classifications). Weight parameters are assigned to each node and randomly initialized. Repeated iterations of backpropagation are performed to optimize a logistic cost function by iteratively taking the partial deriviatve of the weight parameters. Upon the completion of training, untrained features are propagated forward through the network to determine the classification. Initially, I trained a self-coded simple neural network on the gesture data set using the scipy.optimize package and a Truncated Newton optimization algorithm. The simple neural network performed modestly (A: 0.782, F1: 0.776) with optimal conditional parameters of a hidden layer size of 1000 for 1000 epochs with a default value of 1.0 for regularization. After this I utilized the TensorFlow package to train a neural network with two hidden layers using the AdamOptimizer. This model performed the best of all models (A: 0.838, F1: 0.838) with both hidden layers 2000 in size, trained over 1000 epochs with a default value for regularization. I then utilized TensorFlow to train a recurrent neural network using the BasicLSTMCell function. Recurrent Neural Networks utilize a cell which stores and makes reference to a set amount of input data previously considered before making the current classification decision. The model performed only modestly (A: 0.785, F1: 0.761) despite adjusting the number of features considered by the rnn_cell. I then ran a BiDirectional Recurrent Neural Network which segregates input data into smaller sections and considers the classification of the sections before and after it within the input data. Best results were achieved when setting the segregation size to 8 (same as each time segment) however the model performed poorly overall (A: 0.721, F1: 0.716). The results of the recurrent neural networks surprised me and at this point I would look into the integrity of the data itself.

The last model I ran was an ensemble model where the mode classification result for the five best models (K-Nearest Neighbour, Logistic Regression, Simple Neural Network, Deep Neural Network, and Recurrent Neural Network) was used to predict the classifications of the test set. Prior to training any models I would have predicted this approach would of had the best accuracy however it performed only second best (A: 0.825, F1: 0.823).

Final Touches and Recommended Future Steps

I was not entirely satisfied with a model accuracy of 83.8% and in an effort to improve accuracy I then created a data set where the 8 feature values for each time step were converted into a polynomial array. Three models: Random Forest, Logistic Regression, and the Deep Neural Network all performed equivalent or worse on this dataset. In a real world scenario at this point I would look into quality metrics for the data.

Throughout the above process two additional things I noticed suggest that the training data for the gesture set has its deficiencies. The first is that for most algorithms (minus Random Forest), the training data accuracies were around 85-90% which are suprisingly low. Complimenting this, I printed out the details of the model ensemble voting results and there were 96 examples (out of 2397) where all five models mislabeled the classification. This suggests there are numerous training examples within a gesture classification that are vastly different from others.

This is useful knowledge however. If detailed recordings are kept where these extreme examples came from (such as the unit, details of the person who used it, room temperature, etc.) this presents an opportunity to identify under what conditions are data entries particularly erroneous. For example, perhaps it turns out the Myo has difficulty registering gesture 5 for skinny people. With this in mind an additional parameter, such as categorical size of user, could be collected from the user and included in the model to help improve performance.

Lastly, gestures 5 and 6 were the two most difficult gestures to recognize. There may be a possibility to explore amplifying the features responsible for capturing their motion.

Thank you again for reading this analysis. I enjoyed working on the project and look forward to your feedback. If you have any concerns with the program please feel free to contact me.

Stefan
519-871-1811
stefan871@gmail.com

Table 1: Omitted Data Entries

Poor Data Count	Gesture	Example
1	1	310
2	1	434
3	1	978
4	1	1708
5	2	434
6	2	1671
7	2	1708
8	3	434
9	4	434
10	4	471
11	5	434
12	6	434
13	6	550
14	6	1708