

Khayyam Saleem, Zachary Saegesser

Professor Xinchao Wang

CS 559 Machine Learning Fundamentals and Applications -- Final Project

27 April 2018

Performance Analysis of Time Series Classification Techniques

Introduction

Time series classification is a popular machine learning task in contemporary research. Being able to distinguish between various kinds of time series has applications in all disciplines. For our project, we researched and implemented two simple classification algorithms (Naive Bayes, K-Nearest Neighbors) and three advanced algorithms (Support Vector Machines, Convolutional Neural Net, Convolutional Neural Net with LSTM-RNN sub-modules). The experiments were performed over the UCR Lightning-7 dataset, executed over multiple trials with adjusted hyperparameters. Accuracy and other performance metrics for each classifier were collected as well, and we were able to formulate a ranking of how well-suited each classifier was for the task over the dataset.

Dataset

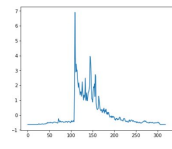
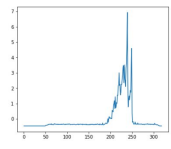
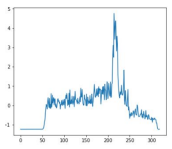
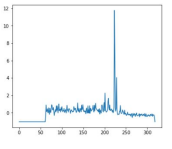
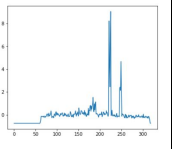
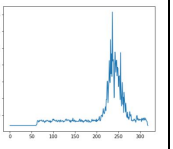
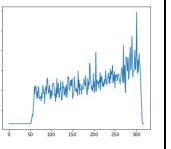
Introduction

For our project, we used a dataset called Lightning7. Lightning7 is time series dataset created in 2002 by a satellite program called FORTE, (Fast On-orbit Recording Transient Events). The FORTE satellite is used to observe and measure short burst events in the atmosphere, thus making it perfect for lightning observations. The satellite is equipped with both radio-frequency and visual instruments allowing the satellite to create spectrograms and power density time series. From the original paper:

“Data is collected with a sample rate of 50 MHz for 800 microseconds ... (A) lightning events, which propagate through the ionosphere (B) trigger the FORTEVHF instruments. The data is then sent to a ground station (C) where scientists preprocess and produce power density profiles.” (Eads, 2002).

Scientists have used lighting events to determine the severity of upcoming weather events. Automation of the classification of all the different kinds of lightning strikes can be an incredibly useful tool for the fields of meteorology and atmospheric sciences.

Classes

Label	CG	IR	SR	I	I2	KM	O
Name	Positive Initial Return Stroke	Negative Initial Return Stroke	Subsequent Negative Return Stroke	Impulsive Event	Impulsive Event Pair	Gradual Intra-Cloud Stroke	Off-Record
# Samples	18	17	14	19	15	38	22
Ex. Time Series							

Class Descriptions

Lightning-7 has seven distinct classes: Positive Initial Return Stroke, Negative Initial Return Stroke, Subsequent Negative Return Stroke, Impulsive Event, Impulsive Event Pair, Gradual Intra-Cloud Stroke, and Off-record (Damian et al.). The first 6 classes are types of lightning strikes and the last class, off-record, represents measurements that were not long enough to be categorized into a class. In total, the dataset in CSV format is approximately 390 KB and is comprised of 143 time series data points, each with 319 observations. The table above shows representatives from each class and their representation in the whole dataset. As one may observe, the classes have little variation between members of the same class but noticeable variation between classes, making our classification task more concrete.

Time Series Classification

Time series analysis is a powerful tool allowing data scientists to study processes and events that can be represented as sequences. Each data point in our experiment was a 319-feature vector, with one observation per time interval. Typical techniques for time series classification include SVM, Genetic Algorithms, hidden Markov Models, convolutional nets, recurrent nets, and many more. For our experiment, we attempted to choose a representative subset of such classification algorithms to execute and rank, so as to present a reasonable spread of the state of the art as it stands. A typical time series breaks down an event into incremental stages ranging from a less than a hundred measurements to over a thousand measurements per sample. Common uses are for forecasting events such as weather and the economy. For example, if a range of humidity observation time series can be classified, than prior to an upcoming weather event, we can classify the humidity observation time series and use that to predict things about the upcoming weather event with the prior knowledge of what event occurred when that class of

time series was last observed. Time series classification, unfortunately, suffers from the “curse of dimensionality.” Since each time step observation is interpreted as a feature of the data point, each data point has high dimensionality, (319 dimensions, in our case), and this makes classification tasks difficult as the algorithm must distinguish between noise and feature. Hence, dimensionality reduction methods become appropriate. For the sake of uniformity in performing our experiments over the different classifiers, no dimensionality reduction such as PCA was performed, and each data point was passed in as it was originally interpreted, sans any feature pre-processing.

Classifiers

Simple Techniques

The first classification algorithm researched was the Naive Bayes Classifier. Naive Bayes is designed on the principles of Bayes Theorem. Bayes Theorem is described as:

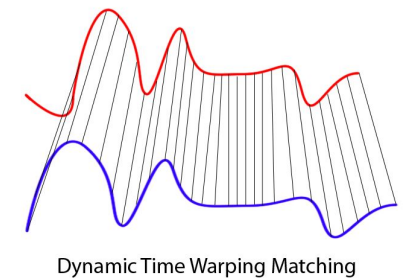
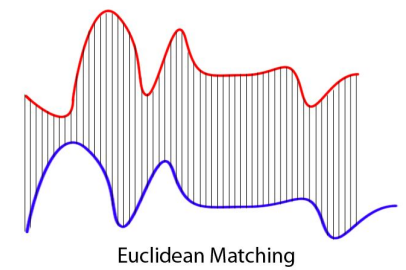
$$P(c | x) = \frac{P(x | c) P(c)}{P(x)}$$

The above equation can be split into four key components: $P(c | x)$ is the posterior probability, $P(c)$ is the prior probability, $P(x | c)$ is the likelihood, and $P(x)$ is the prior probability. The Naive Bayes Classifier assumes independence between features, meaning that each feature contributes to the probability of a sample being in a particular class the exact same no matter what other features are known. The general algorithm for Naive Bayes classifier is to calculate the posterior probability for each feature given a particular class, sum all of the features and compare to the other classes. The class with the highest summation of posterior probabilities is chosen as the assigned class. Naive Bayes is commonly used because it is incredibly simple to implement, has fast prediction time, and performs well with many class predictions. This algorithm is commonly used in recommendation systems because of the natural independence of features in such a system.

For our implementation, we used the mean of a single time series as its representative feature, and assumed that the means within a given class were normally distributed. This decision was made so that minimal feature pre-processing would be required. However, in a more robust implementation, one may wish to extract more representative features such as mean, max, min, variance, and other such metrics to help describe the time series more thoroughly.

The other simple algorithm we investigated for this project was K-Nearest-Neighbors. KNN is a simple classification algorithm, that uses a voting scheme based on the class of the data-points nearest to it to determine its class membership. In normal datasets (non-time series) this is simple: calculate the distance between all the points plotted on the graph, find the nearest k points and the class with the most votes wins the classification. The problem we faced was that our data points are high-dimensional. Additionally, traditional distance metrics do not work in the same fashion on time series as they do on data with lower dimensionality. We needed to

calculate distances in an entirely different way. The simplest solution to this problem is to sum the difference between all points between two time series (Euclidean distance metric).



To improve on our distance computation and make it more suitable for time series data, we used a method called Dynamic Time Warping (DTW). DTW includes the differences in the alignment on the time axis, such as shift of phase. This gives a much more accurate measure for the distance between two time series (Keogh, 2002).

However, with this improved accuracy, we suffer an increase in training time, as DTW is an $O(n^2)$ algorithm. To combat this problem, we implement a locality constraint known as LB_Keogh, a lower-bounding method for time series matching. In our KNN classifier, the LB_Keogh bounding helped us to improve our classification speed without suffering major losses in accuracy.

As time series data is so high in dimension, it has been experimentally determined that a 1-Nearest-Neighbor algorithm is best-suited to accurately classify a particular time series (Xing, 2009).

Advanced Techniques

The first advanced classifier we implemented was a support vector machine (SVM). SVM takes all classes and creates an optimal hyperplane that maximizes the margins between the plane and the classes. This provides a decision boundary for the classification task. We used a one-versus-all classification scheme for our experiment. For time series data, each observation on the time series represents a feature of the data point. Typically, to circumvent the curse of dimensionality, feature reduction or noise reduction is performed on the dataset. However, we chose not to for our experiment to provide a basis for control during our experiments on each classifier. This means that the optimal hyperplane will be in the length-of-the-time-series dimensional space (319, in our case). SVM is often used in time series classification because it allows data scientists to control overfitting. When using SVM classification, one of the most important variables is the kernel. There are many different variations on kernel functions. Our experiment selected just two: linear and RBF. A linear kernel means that the optimal hyperplane does not ‘bend’ or change its shape throughout the space. A nonlinear kernel will ‘bend’ around different classes. Linear SVMs have been known to perform on time series data that has enough inter-class variation, as is the case for ours, but we include the other kernel as well for completeness (Eads, 2002).

Another advanced method we tested was a convolutional neural network (CNN). Convolutional networks are a form of deep learning that use multilevel neural networks to classify inputs. CNN uses hidden layers called convolutional layers. These hidden layers are

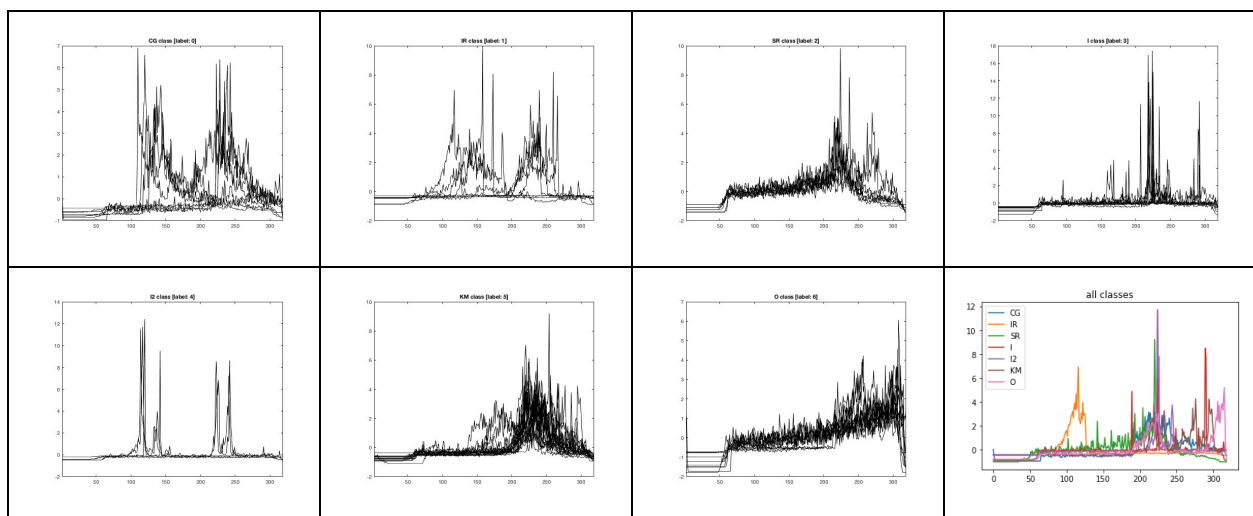
built by sliding a filter box across the input, creating an entirely new layer with higher activation in sections that the filter box of the previous layer was designed to identify. This is repeated as the CNN is trained, adjusting the filters for each training run. This method is incredibly useful for pattern detection, lending to its most common use in image and text pattern recognition and other similar areas. The intuition for the usage of CNNs on images carry over to time-series data well. The main difference in the code is the stride argument we pass to the convolutional layer, as rather than striding UP the image as would be the case for an image classification problem, we slide ALONG the time series.

We also examined another special kind of convolutional net specialized for time series classification, an LSTM Fully Convolutional Network (Karim, 2017). This method is extremely new, and involved augmenting the fast classification performance of Temporal Convolutional layers with the precise classification of Long Short Term Memory Recurrent Neural Networks. This method is able to take advantage of the comprehensive nature of convolutional networks while retaining the “memory-aware” property of recurrent neural networks. Our particular implementation is even more recent, and involves augmenting the existing univariate time series classification models, LSTM-FCN and ALSTM-FCN, with a squeeze and excitation block to further improve performance. This is an implementation of the Jan 2018 paper cited (Karim, 2018).

Experimentation and Results

Data Observation

We began with an inspection of each of the classes in our dataset, to determine the spread of the data and the degree of inter- and intra- class variation. The classes are plotted below.



Naive Bayes

The Naive Bayes classifier, as mentioned earlier, assumed that the intra-class time series means were normally distributed. Using the Gaussian Naive Bayes classifier library from scikit-learn in Python, we were able to achieve an accuracy of 0.573. This was conducted with a train-test split of 70-73.

1-Nearest Neighbor

With the LB_Keogh localization technique, we were able to reduce our training and classification time from 1m 32.5s to 0m 48.3s. We computed performance metrics using the Euclidean distance metric and using the Dynamic Time Warping distance metric. These experiments were also conducted with a 70-73 train-test split. Notably, this classifier performed better than some of the advanced classifiers.

EUCLIDEAN

	precision	recall	f1-score	support
0.0	0.75	0.60	0.67	10
1.0	0.20	0.11	0.14	9
2.0	0.56	0.83	0.67	6
3.0	0.25	0.14	0.18	7
4.0	0.50	0.30	0.37	10
5.0	0.59	0.84	0.70	19
6.0	0.71	0.83	0.77	12
avg / total	0.54	0.58	0.54	73

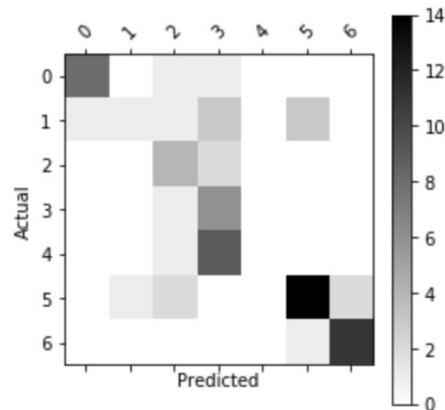
DYNAMIC TIME WARPING

	precision	recall	f1-score	support
0.0	0.73	0.80	0.76	10
1.0	0.80	0.44	0.57	9
2.0	0.50	0.67	0.57	6
3.0	0.46	0.86	0.60	7
4.0	1.00	0.20	0.33	10
5.0	0.77	0.89	0.83	19
6.0	0.92	0.92	0.92	12
avg / total	0.77	0.71	0.69	73

Support Vector Machine

The dictionary shown below is a mapping from classes to number of correct classifications. We can see from here that class 5 had the highest number of correct classifications, but it also had the highest representation in the dataset so the accuracy may be skewed. Additionally, we can see that classes 1 and 4 had very low accuracies. We hypothesize that this is due to their relatively low representation in the dataset as well as their lower inter-class variation. These two classes had less distinguishing features than some of the others with higher classification rates. A confusion matrix is also shown. With a train-test split of 70-73, and using a linear kernel function, we were able to get an accuracy of 0.6027.

{0: 8, 1: 1, 2: 4, 3: 6, 4: 0, 5: 14, 6: 11}



Convolutional Neural Network

For our basic convolutional classifier, we borrowed from a model used for human activity recognition, making relatively few changes. The architecture includes three convolutional layers, with batch normalization between each and softmax activation functions in the fully-connected layers. This classifier, again, was trained with a 70-73 train-test split, and the accuracy on the validation data over 3000 iterations was 0.639.

LSTM Fully Convolutional Neural Network

This last classifier is extremely new, but gave us our best results. It provided a significant improvement over all the other classifiers, but also took the longest to train at 24m 33.4s. It was implemented using the Keras library with TensorFlow backend in Python, and yielded a max accuracy of 0.8904.

FINAL RANKING

LSTM FCN: 0.8904, 1NN with DTW: 0.7700, CNN: 0.6390, SVM: 0.6027, GNB: 0.5730

Bibliography

- Damian R. Eads, Daniel Hill, Sean Davis, Simon J. Perkins, Junshui Ma, Reid B. Porter, James P. Theiler, "Genetic Algorithms and Support Vector Machines for Time Series Classification," Proc. SPIE 4787, Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation V, (6 December 2002);
- Karim, F., Majumdar, S., Darabi, H., & Harford, S. (2018). Multivariate LSTM-FCNs for Time Series Classification. CoRR, abs/1801.04503.
- Karim, Fazle & Majumdar, Somshubra & Darabi, Houshang & Chen, Shun. (2017). LSTM Fully Convolutional Networks for Time Series Classification. IEEE Access. PP. 10.1109/ACCESS.2017.2779939.
- Ratanamahatana, Chotirat Ann, and Eamonn Keogh. "Making Time-Series Classification More Accurate Using Learned Constraints." Cs.ucr.edu, University of California - Riverside, 2002.
- Shmueli, Galit. Smoothing 2: Moving Average for Forecasting. National Tsing Hua University, 30 Nov. 2016.
- Xing, Z., Pei, J., & Yu, P. S. (2009). Early prediction on time series: A nearest neighbor approach. In IJCAI International Joint Conference on Artificial Intelligence (pp. 1297-1302)