Penn Project Report
Machine Learning in Real Time Character Recognition

Abstract

This paper presents a MEMS based digital pen for handwriting recognition. While there are several devices which are capable of digitizing handwritten notes, all commercially available devices require secondary hardware to function. We propose a single device solution for digitizing text as it is written on a page. By calculating the dynamics of the pen from IMU measurements, we are able to estimate the pen tip accelerations and rotations as a function of time. We evaluated SVCs, decision trees, Ada-Boost, and logistic regression learning algorithms based on their ability to recognize handwritten characters and achieved 96.2 % accuracy using an SVC with a lineal kernel.

1. Prototype

In this section we give a brief summary of the hardware incorporated into the proposed device. We first describe the mechanical device before then discussing the electronics.

1.1. Hardware

The pen layout and body was designed in CAD (pictured below) to ensure appropriate packing of components. The clear body was 3D printed in ABS in the University's prototyping labs.
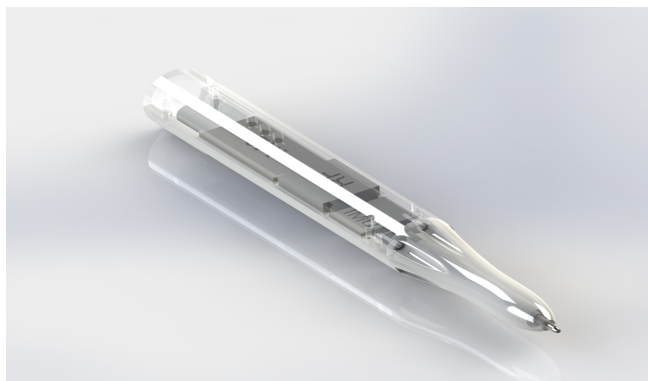
Figure 1. Rendering of the 3D printed model of the penn

1.2. Electronics

An ATMega32u4 microcontroller provides on-board processing and interfaces to the wireless module, IMU, switch, and indicator LEDs. Two Nordic nRF24LE1 transceivers handle wireless communication between the pen and base station computer. Inertial measurements are provided by an MPU6050 digital IMU with configurable gain triaxial accelerometers and gyroscopes. A Honeywell HMC5883L triaxial magnetometer provides a reference to earth's magnetic field for orientation. A switch behind the pen tip detects contact with the page for character segmentation. Three LEDs on the case display pen state. The pen also includes on-board batteries, power management, and an on-off switch to track idle and writing time of the pen, which translates in useful and un-useful data.

1.3. Filtering the data

The raw data from the IMU pose issues for learning algorithms. Sources of variance can include posture, handedness, roll angle of the pen, and sensor drift. To minimize sources of error, we compute the trajectory of the pen tip on the page to generate features in a four step process.

1. IMU measurements are scaled to engineering units and are low-pass filtered to remove high frequency noise from vibration.

2. We use Madgwick's open source Kalman filter and at-

titude estimator to find the orientation of the pen relative to the page.

3. We transform the IMU measurements into the ground frame relative to the page and double-integrate the acceleration to estimate the displacements of the pen tip.

4. Finally, the position and velocity estimates are high-pass filtered to remove drift.

From this IMU data processing we obtain the x,y,z position, velocity, and acceleration of the pen in the page frame as well as the euler angles and angular rates of the pen in the page frame at 300Hz sample rate.

Down below we can see how the user interface for testing looks like, and an example of the trajectory reconstruction of a "g" letter. In the MATLAB plots we can see all the readings of the IMU in real time, the attitude estimation of the penn in 3D (Which corresponds to the real position of the penn with a very slight delay), and the real time trajectory reconstruction of the writings with other parameters.
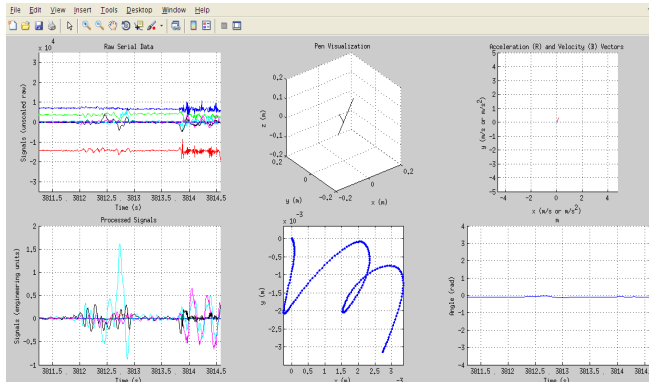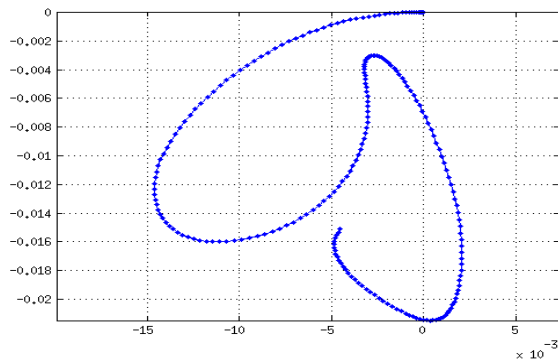

Figure 2. Writing interface for character recognition

Figure 3. Recovered letter "g"

2. Learning

2.1. Features

To generate our feature vector, we find the mean, standard deviation, and RMS of each of the signals x, y, z position, velocity, and acceleration and euler angles and angular rates, then interpolate each of the signals to a uniform length. Our feature vector consists of the character writing time, these interpolated signals, and their mean, standard deviation, and RMS. This generates a feature vector of length 15*N + 46 where N is the interpolated signal length. After feature vector generation we standardize our features.

2.2. Learning Algorithms

We first tested Support Vector Machines with different Kernels to compare the results we were getting from them. This ones were all runned with 5 folds.

Algo.	Kernel	param	Sig Len	Time(s)	Acc.
SVC	linear	-	10	2.114	96 %
SVC	poly	2	10	2.07	91 %
SVC	poly	3	100	6.681	91 %
SVC	gaussian	5E-1	10	5.057	3 %
SVC	gaussian	5E-4	10	2.124	77 %
SVC	gaussian	5E-5	10	3.05	29 %
SVC	gaussian	5E-4	100	11.357	93 %
SVC	gaussian	5E-5	100	13.709	82 %
SVC	gaussian	5E-3	1000	309.19	3 %

As we can clearly see, while the linear and polynomial kernel do a pretty good job to classify the characters, the rbf is VERY prone to subtle variations in the parameters, which makes it a non robust method for character recognition. The gaussian kernel smoothes everything if you don't have the correct parameters, and that is why it's a non reliable method for this application in particular. We can see how the performance for the gaussian kernel goes from 3 to 77% and then back down to 10% depending on the choice of gamma. And then once again changing the signal length we get very different performances. On the other side, we can see that the accuracy obtained by the polynomial kernel is not even affected by the degree of the same and the variations are little while it runs very fast. Beer then, we get the best performance with the linear kernel, where we don't have to worry about the gamma or the polynomial degree.

As expected, decision trees perform poorly on this dataset. Decision trees in general have lower predictive power than the other methods applied here. Furthermore, we expect the ideal decision function to be much smoother non axis-aligned than the axes-parallel hyper-rectangles of decision trees can produce. This is because the integrated time series features are highly correlated with one another. Despite the

smoothing of AdaBoost, AdaBoosted decision trees still underperformed other methods for some settings, and got over 97% accuracy for some other setting, but taking a lot of extra time that is essential for the quickness of writing and 3% increase in performance does not justify the speed of AdaBoost at this settings.

Logistic regression performs well achieving accuracy, precision, and recall of 89% with both l1 and l2 norm distance metrics.

Several letters are very similar in stroke trajectory. All learning algorithms struggled with these sets of letters more than other sets of letters as the confusion matrices reveal. Similar letter pairs include o and 0, p and b, 9 and q, y and x, h and n, 0 and 6, 5 and 6, 7 and 8, v and r, m and k, and a and u. Some of these letter pairs look visually different from each other, but are composed of similar stroke patterns (for example a script k and an m share the same vertical beginning stroke, direction reversal, loop and second direction reversal).

3. Performance

3.1. Tested Algorithms

The algorithms that were tested were SVCs with different kernels, Decision Trees, Logistic Regression with L1 and L2 penalties, and AdaBoost decision Stumps. While all of the performed with a lot more than random success, there were differences between them.

3.2. Final Performance

As we can well see, even though the logistic regression gets 89% accuracy, the ones that do better are the SVCs with linear kernel. SVCs with linear kernels present a robust machine learning algorithm that is not going to be changed completely if we add a new feature to the feature vector for example. They perform quickly, efficiently, and accurately making them the best algorithm for the task. We will next compare some of the results between the SVCs with linear kernels and the Decision trees that among the ones that perform poorly, still get over 60% accuracy.

We can observe how the true positives in the SVCs go steeper and faster into the true positive area, making less mistakes in all instances.
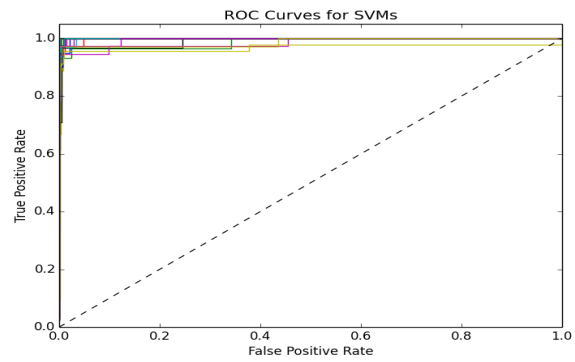

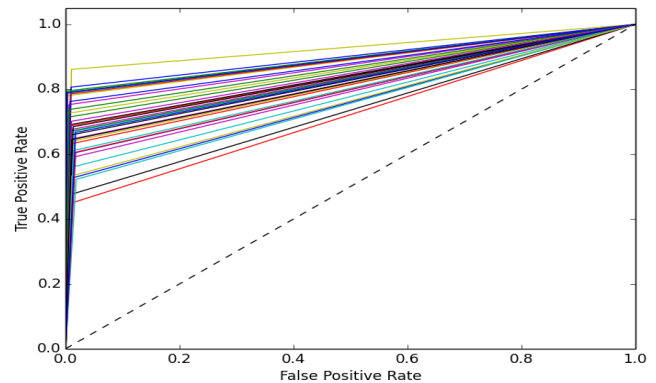*Figure 4.* ROC curves of an SVC with linear kernel

Figure 5. ROC curves of a Decision Tree with max depth = 100

Next we will be looking at the confusion matrices of SVCs with linear kernel and Decision Trees. We will appreciate this index in the way that it visually and automatically compares the accuracy of a method to another, and which are the characters that are most commonly mis-classified.
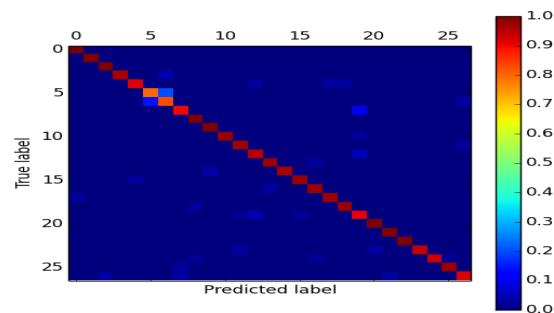

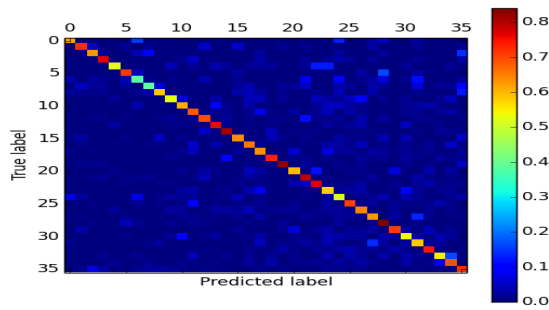*Figure 6.* Confusion Matrix of an SVC with linear kernel

Figure 7. Confusion Matrix of a Decision Tree with max depth = 100

We can observe in these confusion matrices how the diagonal is much stronger than the sides, meaning that most of the instances of a certain class are classified correctly, and that there are very little false positives and false negatives. And it is visually clear how different the performance for these two methods is, because we can see a lot more light blue spots in the false positive area in decision trees for things that were mis-classified.

4. Results

Here you can see a video of our interface, readings, character tracking, and an example of a "Hello World" recognition: `https://www.youtube.com/watch?v=L0AaqIswKBE&list=UUEdPimfOpE81_NbsL7sXQhA&spfreload=10`

Acknowledgments