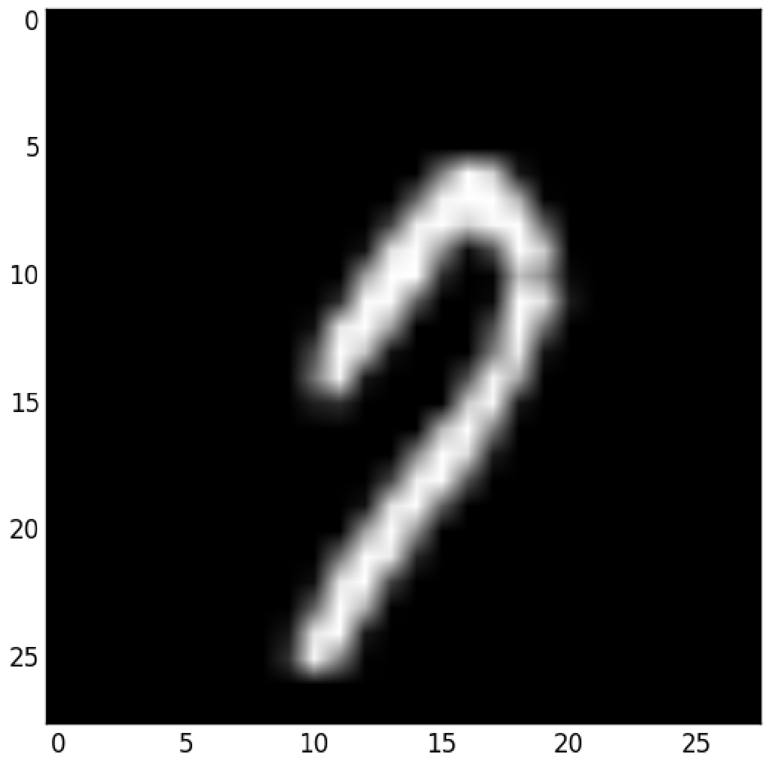
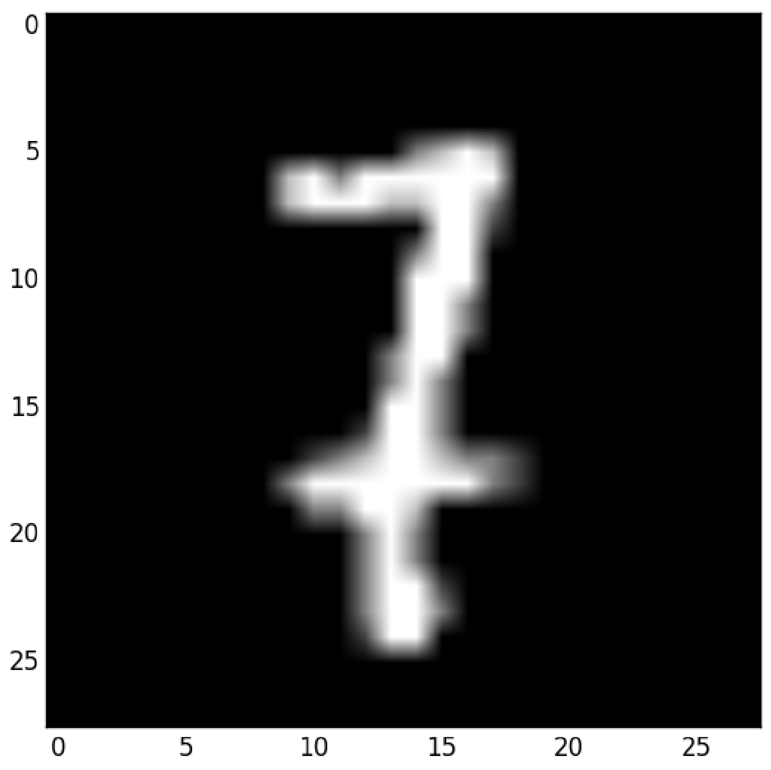
Corey Grief and Adam Snyder
Bryan Pardo
EECS 439 Machine Learning
3 December 2015
Homework 9

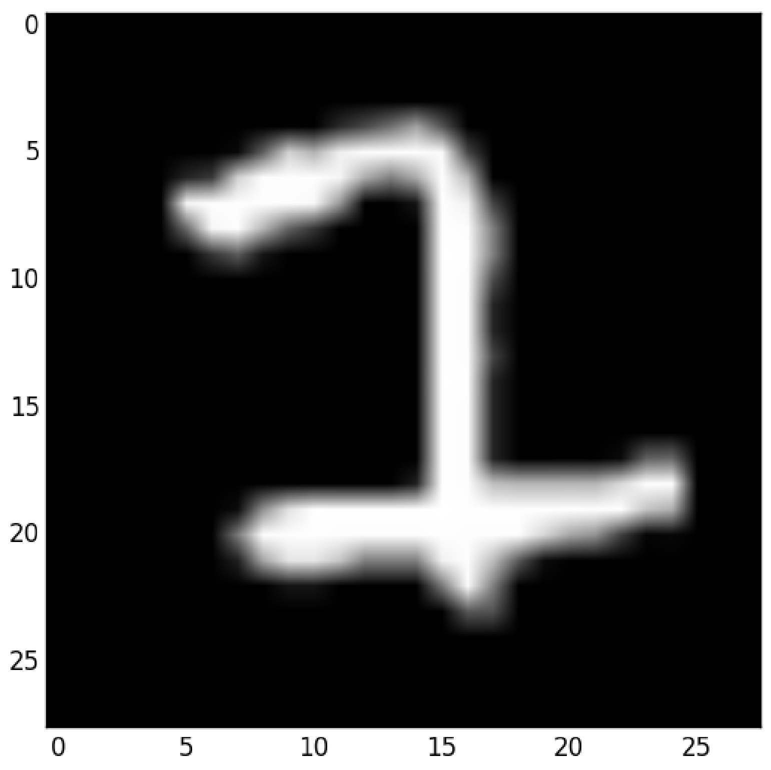
Part 1

A. total: 60,000 0: 5,923 1: 6,742 2: 5,958 3: 6,131 4: 5,842 5: 5,421 6: 5,918 7: 6,265 8: 5,851 9: 5,949

B. We looked through examples of the number 7 and found these to be shocking.







These examples might be challenging because they appear to be other numbers, like 9 or some versions of 1.

C. We chose our training and testing sets randomly without replacement. For a classifier to generalize over unseen data, we want the training and testing sets to be representative of the population of interest.In a perfect world, we would look through all of the data in order to get examples that both cover all of the variations in the examples, and are representative of the data as a whole. However, this in infeasible due to the size of the data set. Therefore, we chose to randomly draw from the examples without replacement in order to get a set that is approximately representative. Our training and testing sets are disjoint, and training set is 1000 of each digit, for 10000 total images. The testing set is 100 of each digit for 1000 total images.

D. The images are converted into a vector for classification where each pixel is a feature. It is important that similarly labeled examples line up so that their feature vectors appear similar. Because of the way the data set is structured, the classifier is actually learning which pixels are on or off and the correlation between pixels over the examples.

Part 2

Naive Bayes

A. Naive Bayes Classifiers are simple probability-based classifiers that apply Bayes' theorem with the assumption of independence between features given the class variable. The classifiers consider each of the features to contribute independently to the probability of an example falling within a given class. This assumption causes such classifiers to be quite simple and efficient and only requires a small amount of training data.

Features on the digit data for this classifier would be each pixel in the image, which can take a value between 0 or 1 depending on that pixel's intensity. If the pixel has an intensity greater than .5, it is 1, otherwise it is 0. This kind of data obviously violates the independence assumption of the classifier, but naive bayes often performs well even when this assumption is violated. The classifier will output a class corresponding to the digit that the classifier classified the input as.

B. Hyperparameters for Bernoulli naive bayes:

- **alpha**: Additive smoothing parameter. This attempts to smooth categorical data by adding a certain baseline to each feature, such that even features that are not observed have some small probability. (e.g. black swan discussion). 0 is no smoothing. Higher numbers increase the smoothing applied to each feature.
- **binarize**: threshold for converting features to booleans. Allows for changing what the pixel intensity threshold is for marking it as "on" vs "off"

Support Vector Classification

A. A support vector machine creates a binary classifier by mapping example inputs into a higher dimensional space using a kernel function and then finding the hyperplane that separates the examples into two groups such that the margin between the groups is maximized. To use it to classify images, the data would be encoded as a list of binary digits representing each pixel in the image as being either on or off. The SVM would interpret each pixel as a dimension. The classifier will output the class of the predicted value of the test image.

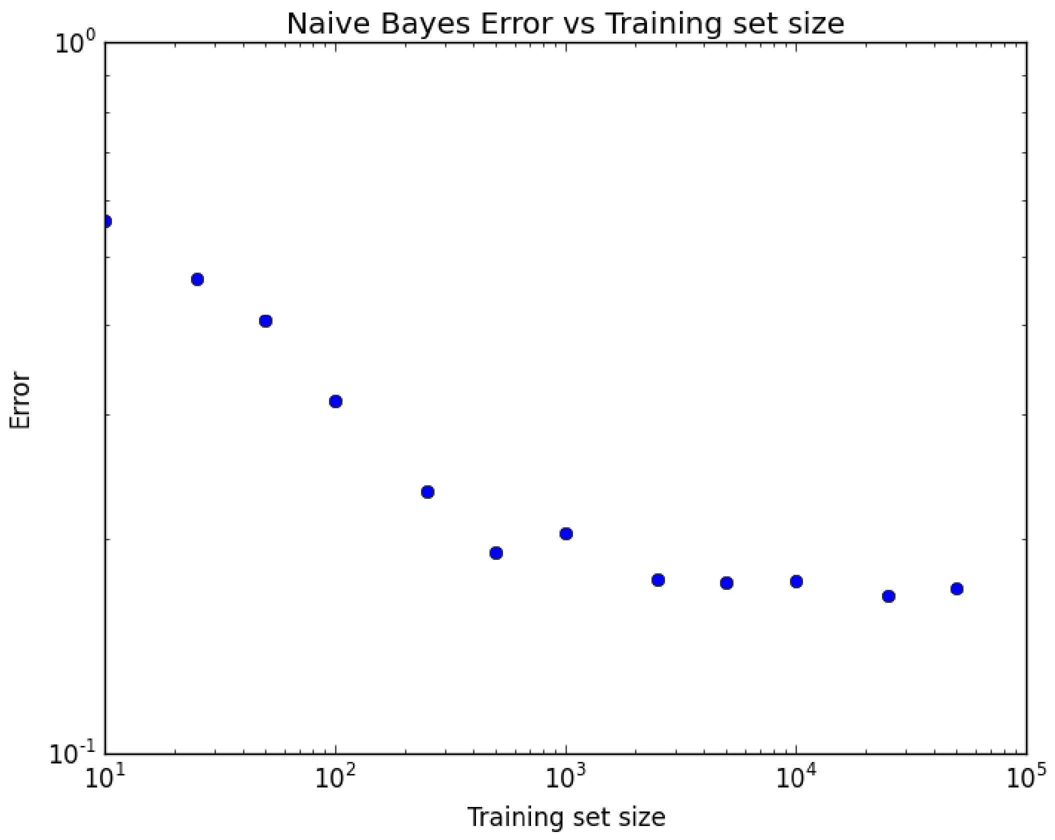
B. Hyperparameters for SVC:

- **kernel**: the kernel function to use, which changes how new training data is mapped to a higher dimension space
- **C**: the cost of classification. A large C leads to low bias and high variance, while a small C leads to high bias and low variance.
- **gamma**: kernel coefficient, which controls the sharpness of the peaks where the points are raised. A high gamma gives a soft bump, resulting in high bias and low variance, while a low gamma gives a sharp bump, resulting in low bias and high variance.

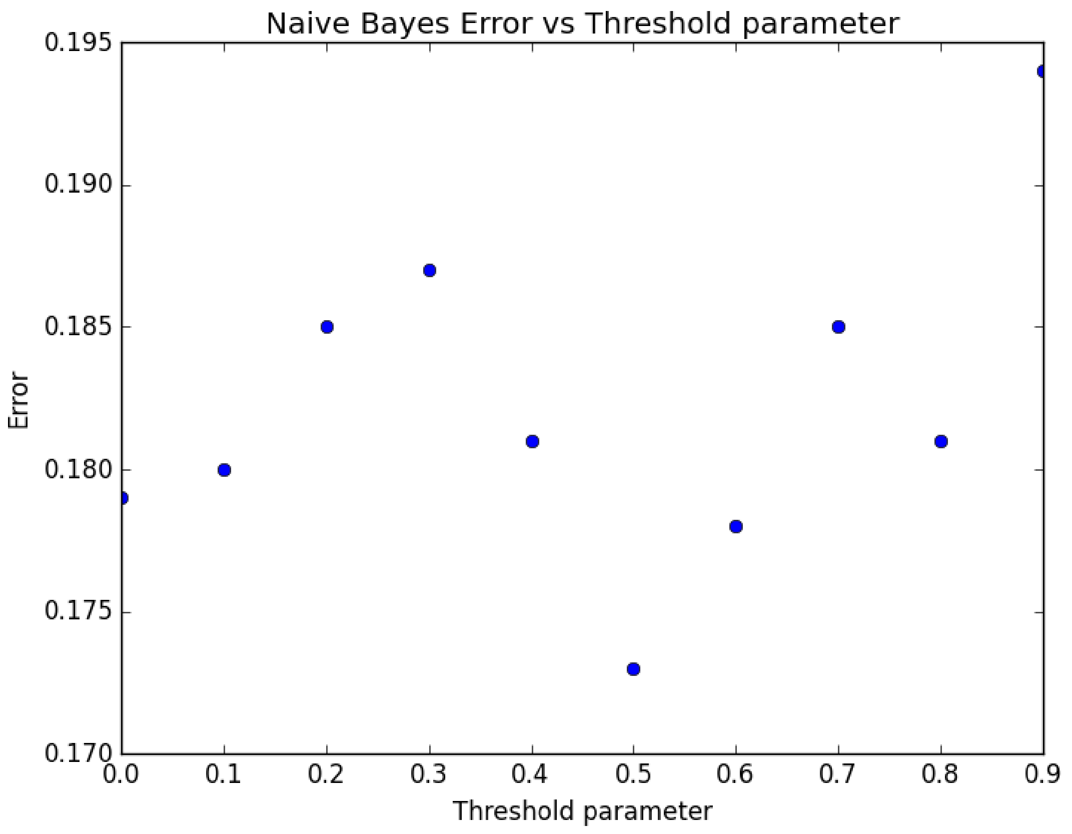
Part 3

B.

Alpha = **1**, Threshold = **0.5**:

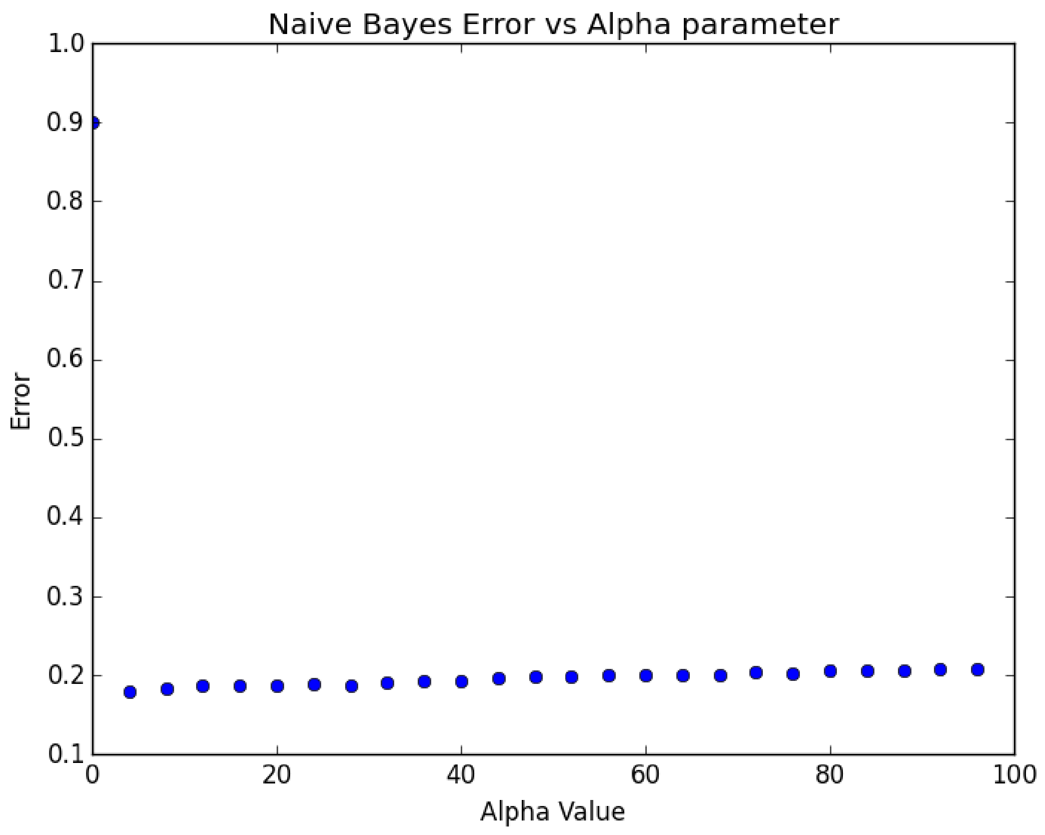


training_set_size = **10000**, Alpha = **1**:



There does not appear to be a strong correlation between threshold for binarizing the data and error rate. Values in the range [0.4, 0.6] appear to be slightly better than the rest.

training_set_size = **10000**, Threshold = **0.5**:



As you can see, apart from alpha = 0, alpha does not have a major effect on error rate. In fact, increasing alpha slightly increases error rate.

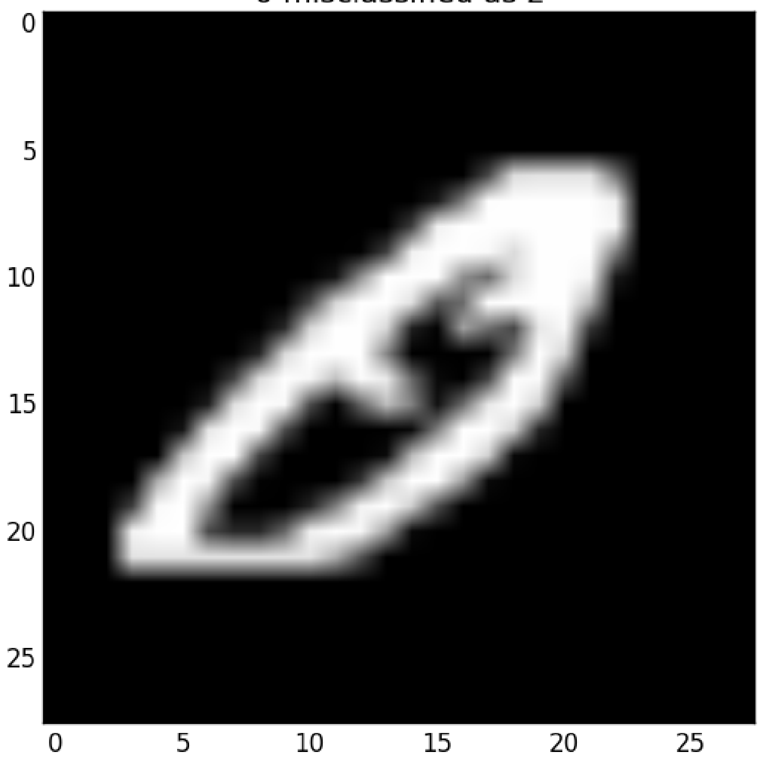
Based on these results, the optimal parameters chosen were Alpha = 1, Threshold = .5 Optimal error rates of ~16% error.

Confusion Matrix

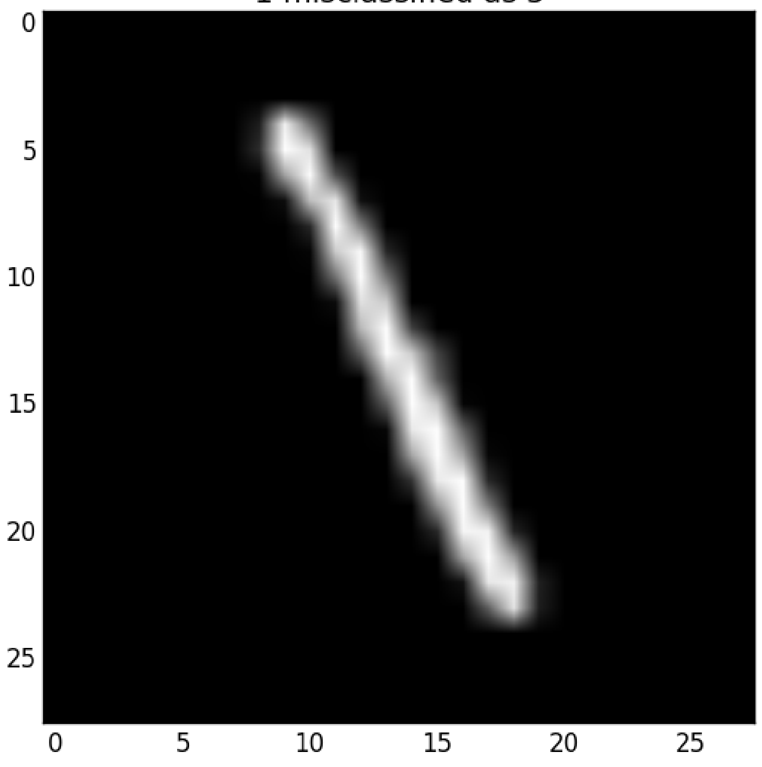
```
              Classifcation
         0  1  2  3  4  5  6  7  8  9
      ------------------------------
    0 | 90  0  1  1  0  4  0  0  4  0
 E  1 |  0 97  0  0  0  1  1  0  1  0
 x  2 |  4  3 79  7  1  0  1  1  4  0
 p  3 |  1  1  4 78  0  2  1  4  7  2
 e  4 |  1  1  0  0 77  0  0  0  2 19
 c  5 |  3  2  2  8  6 69  0  2  5  3
 t  6 |  0  1  5  0  1  2 90  0  1  0
 e  7 |  1  2  0  0  5  0  0 85  0  7
 d  8 |  1  6  0  5  2  3  1  0 77  5
    9 |  1  2  0  0  3  0  0  5  4 85
```

C.

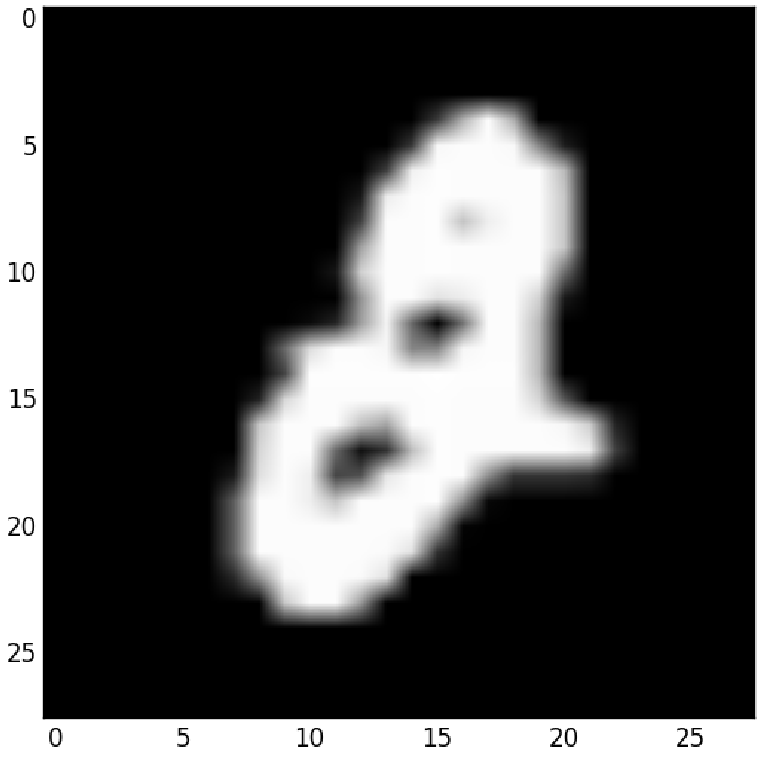
0 misclassified as 2



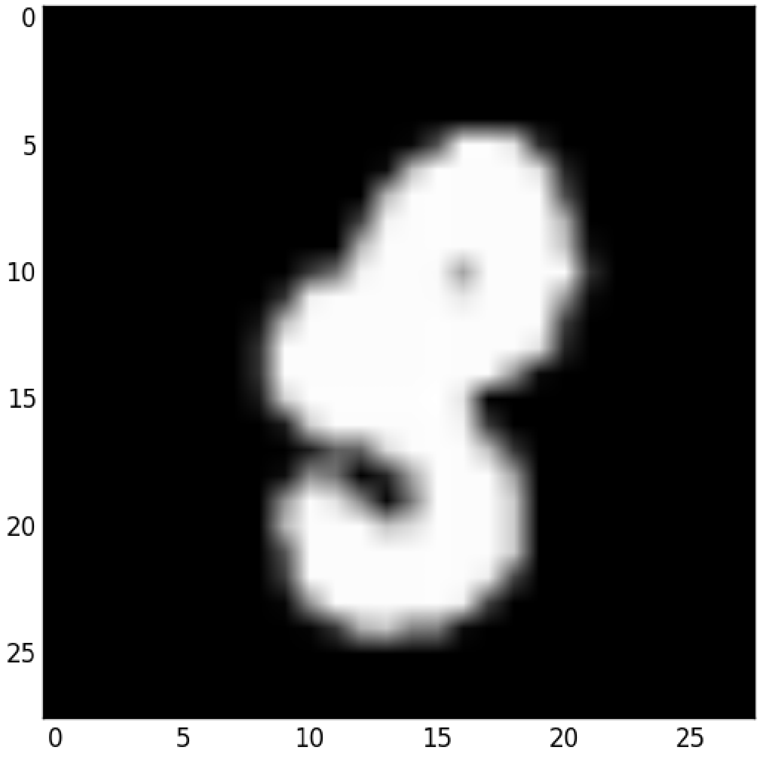
1 misclassified as 5



2 misclassified as 8



3 misclassified as 8

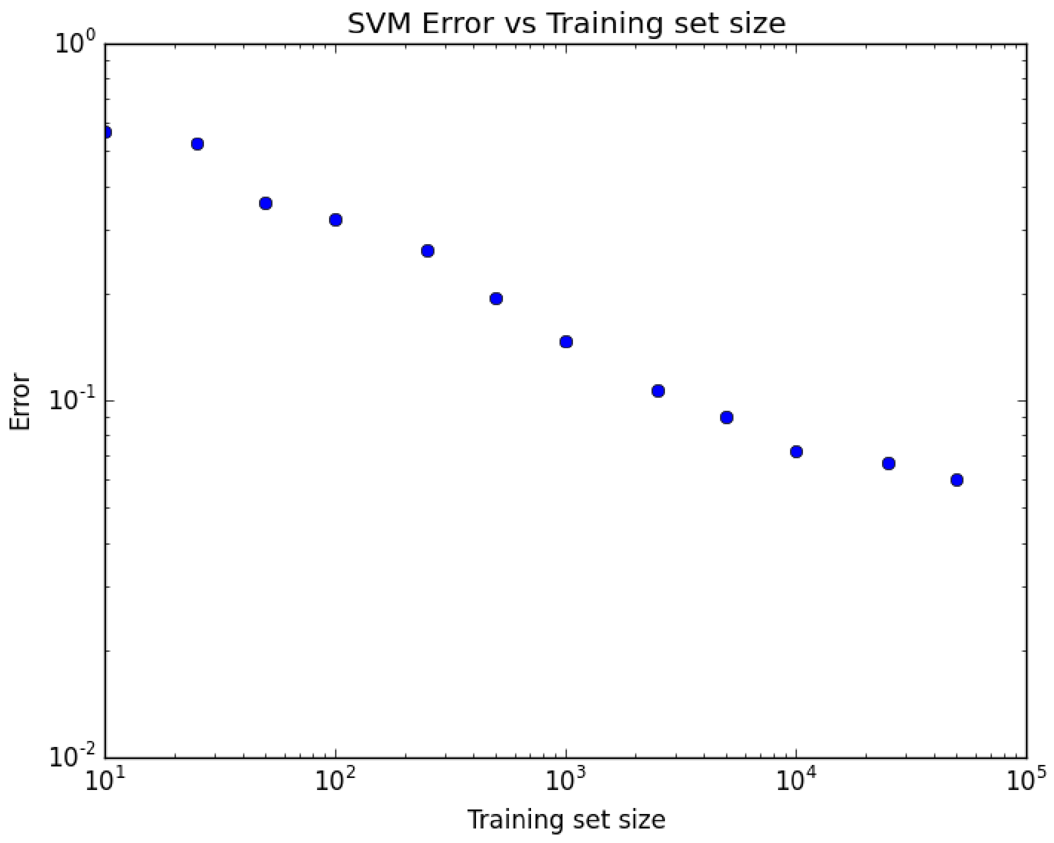


The above misclassified examples, are unclean(0, 2), resemble other numbers(3), or are clean but still misclassified (1) The most often errors occur from unclean images or inconsistent number styles leading to misclassification, especially when some parts of numbers share similar regions. Because the classifier only compares activated pixels and not more complex features such as strokes, empty spaces, etc., digits with similar sections of pixels, like 4 and 9, where they share most of the activated pixels, often result in error.

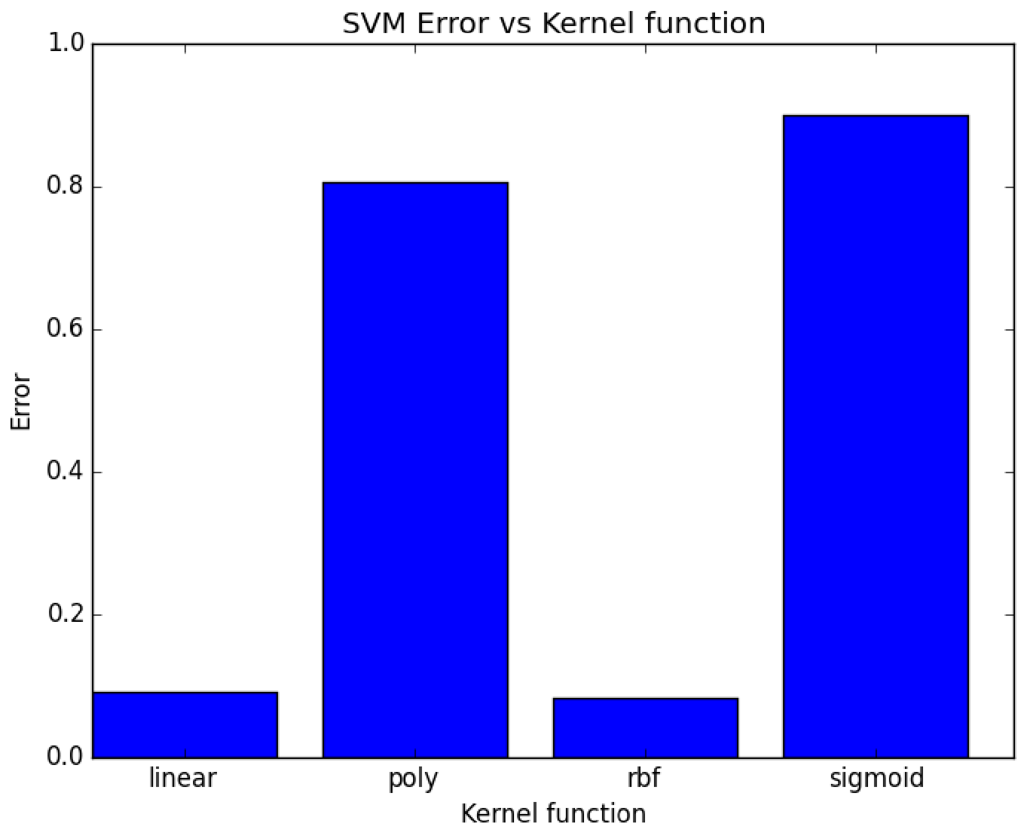
Part 4

B.

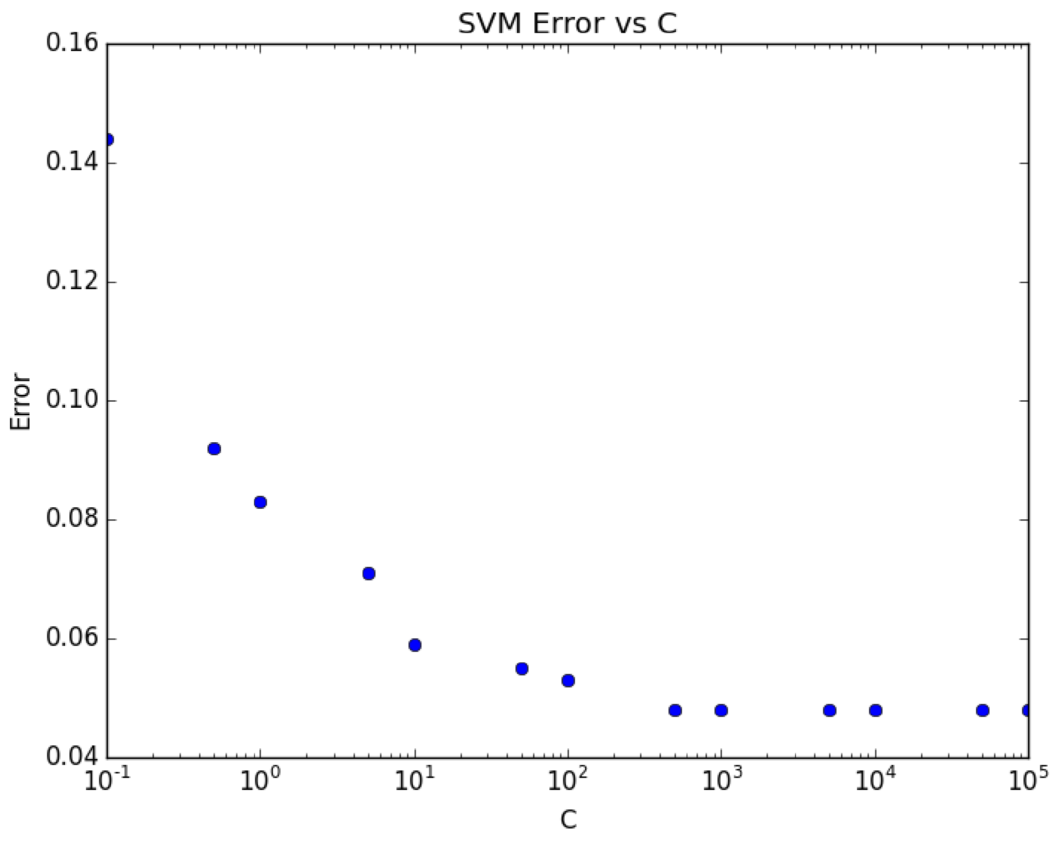
kernel =**'rbf'**, C = **1**, gamma = **'auto'** :



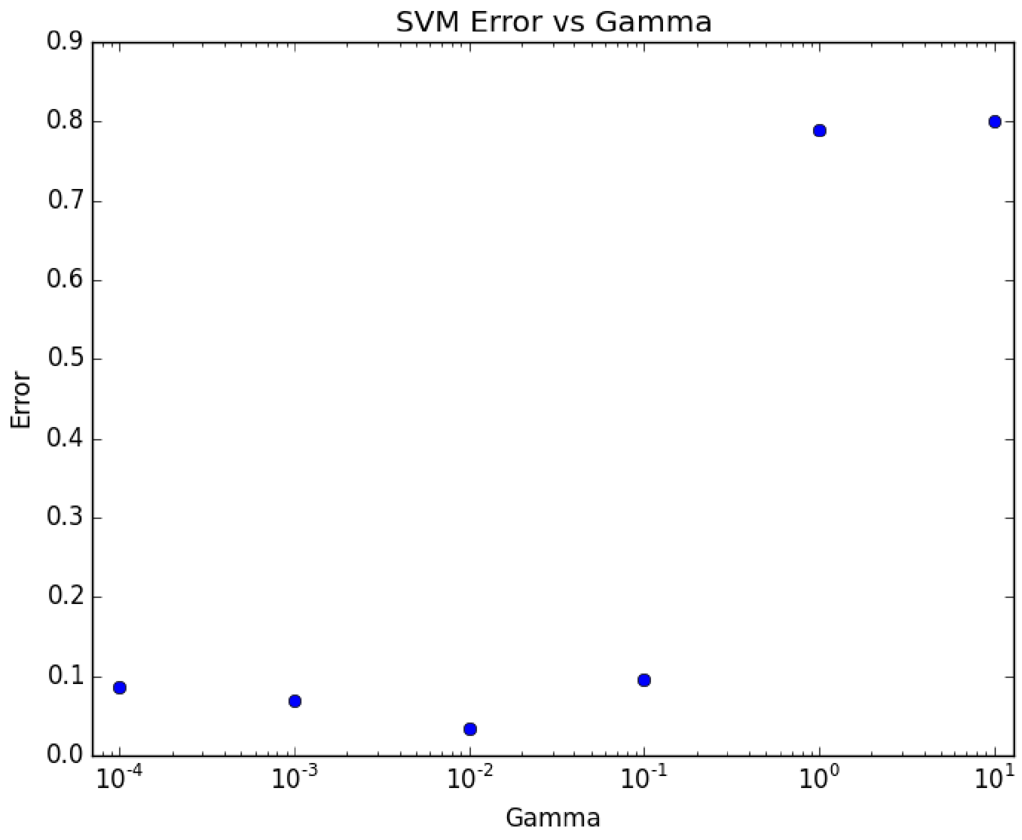
training_set_size = **10000**, degree = **3**, C = **1**, gamma = **'auto'** :



training_set_size = **10000**, kernel = **'rbf'**, gamma = **'auto'** :



training_set_size = **10000**, kernel = **'rbf'**, C = **8** :



Based on my results, the default RBF kernel function performs the best. Error is further minimized by increasing the C value, which corresponds to the cost of classification. A large C leads to low bias and high variance, while a small C leads to high bias and low variance. Finally, the gamma value can be tweaked to optimize accuracy, with a value on the order of 0.1 being most optimal. Gamma affects how much influence each training example has.

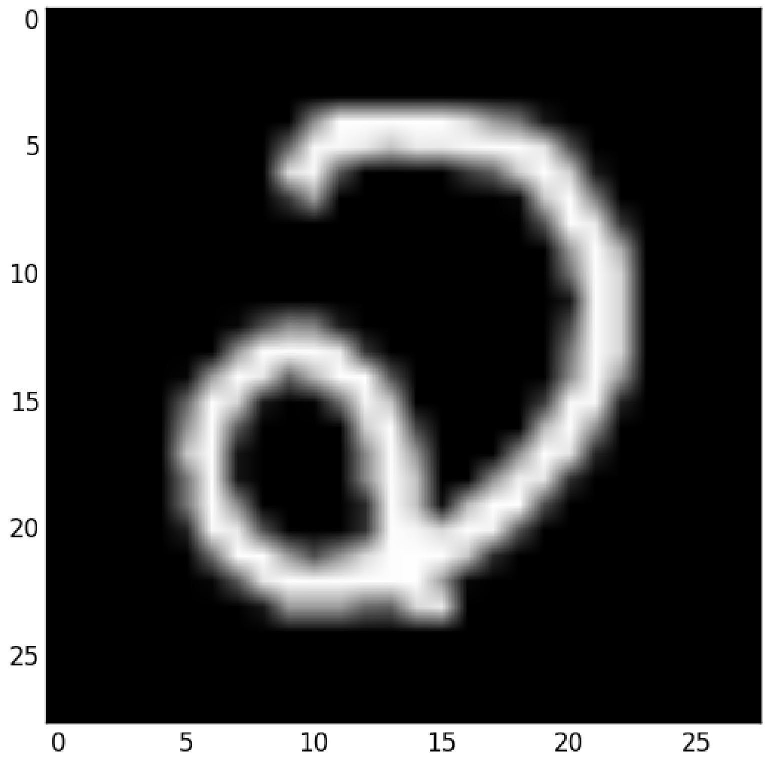
After a performing grid search to optimize the parameters further, the best SVM classifier had hyperparameters kernel = **'rbf'**, C = **8**, and gamma = **0.02**, and run with a training_set_size = **10000** had error = **0.029**.

Confusion Matrix:

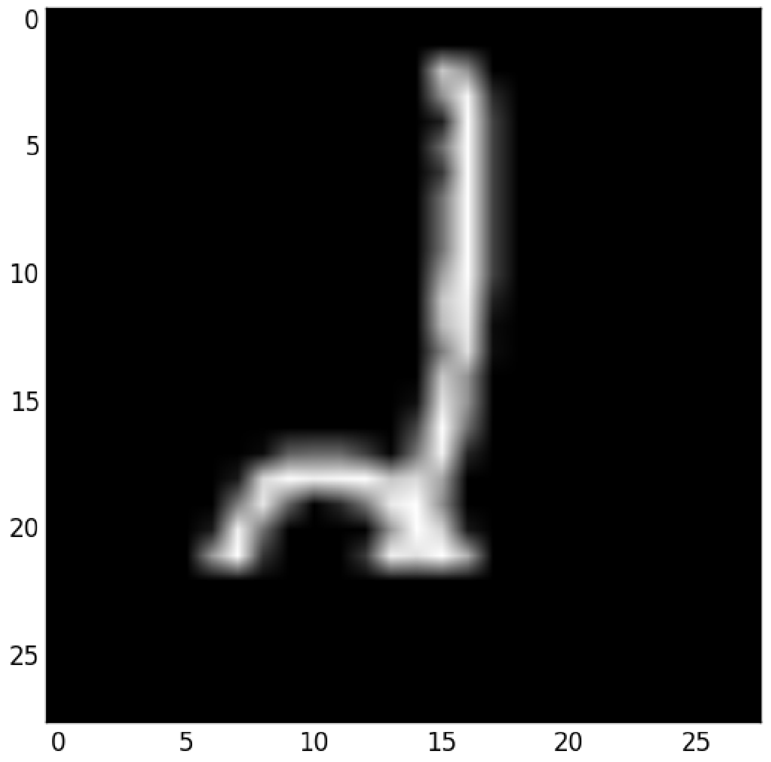
```
              Classifcation
           0  1  2  3  4  5  6  7  8  9
         ------------------------------
      0 | 99  0  0  0  0  0  1  0  0  0
   E  1 |  0 99  0  0  0  0  0  1  0  0
   x  2 |  3  1 92  0  2  1  0  1  0  0
   p  3 |  0  0  0 98  0  0  0  1  0  1
   e  4 |  0  1  0  0 98  0  0  0  0  1
   c  5 |  1  0  0  2  0 96  0  0  0  1
   t  6 |  0  2  0  0  0  0 98  0  0  0
   e  7 |  1  0  0  0  0  0  0 99  0  0
   d  8 |  0  1  1  1  0  0  0  0 97  0
      9 |  0  0  1  2  2  0  0  0  0 95
```

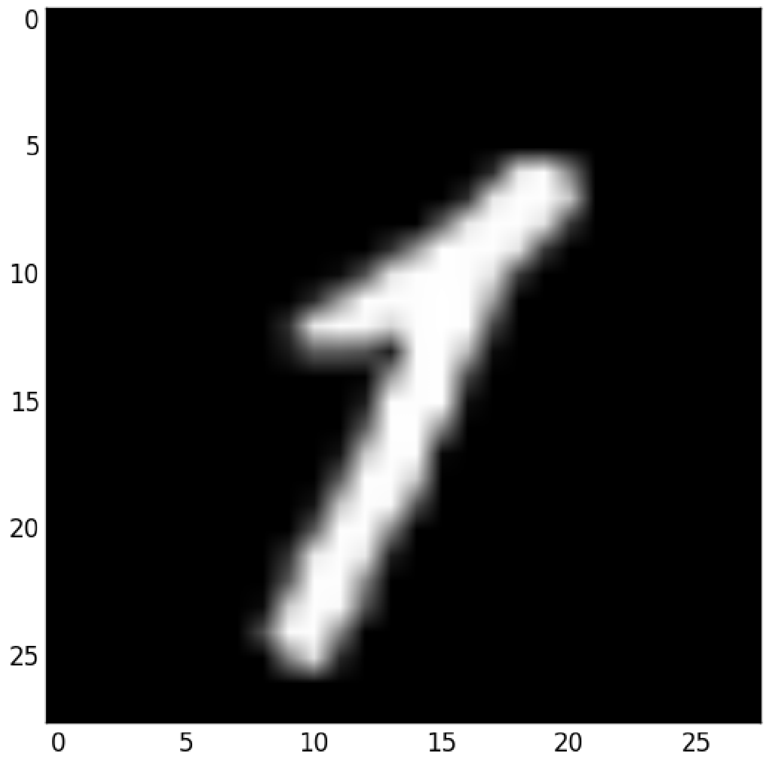
C.

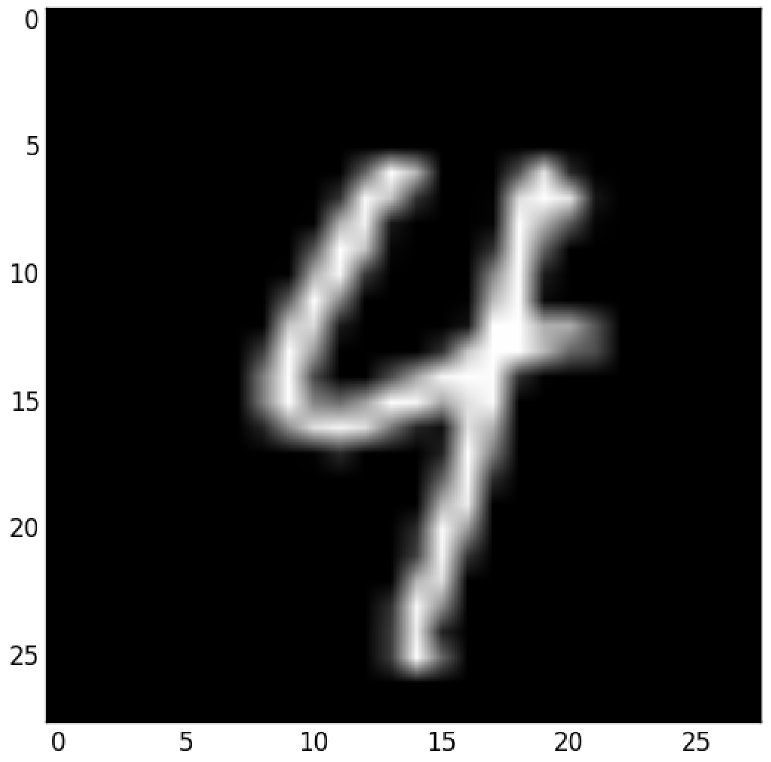
2 misclassified as 0



2 misclassified as 1




1 misclassified as 7


4 misclassified as 9

The misclassified examples were a mix of poor examples, clean examples, and examples that were predicted to fail from part 1B. As such, we have learned that not all of the examples are perfect, but since the SVM treats each pixel as a feature instead of looking at aspects of the overall shape of the image as features, images like 2, which shares many pixels with the 0, are misclassified. The fact that there is a gap in the stroke is not factored into the classification.

Part 5

The Support Vector Machine outperformed the Naive Bayes classifier remarkably, with ~13% reduction in overall error rate. Misclassifications occured for almost any pair of images using the Naive Bayes classifier, while the Support Vector Machine mainly had mis-classifications for difficult examples.

One reason the SVM may have had better performance is because the Bayesian Independence assumption is not met by this feature representation. Pixel values are not independent of one another given a class by nature of how handwriting works. This violation of the fundamental assumption of a Naive Bayes classifier most likely contributes to some of the error.

Part 6

A. error: **0.318**
Confusion Matrix:

```
              Classifcation
         0  1  2  3  4  5  6  7  8  9
       ------------------------------
    0 | 86  0  1  1  1  3  3  1  4  0
 E  1 |  0 89  5  0  0  1  0  1  4  0
 x  2 |  3  2 49  4  3  1 21  4 12  1
 p  3 |  3  1  1 70  1 12  1  0  8  3
 e  4 |  1  2  2  0 68  2  0  4  3 18
 c  5 |  8  1  2  9  2 55  2  6  8  7
 t  6 |  4  1 12  1  2  1 76  0  3  0
 e  7 |  0  1  2  1  6  0  0 76  2 12
 d  8 |  2  1  2  6  2  6  1  0 71  9
    9 |  0  1  2  1 39  2  1  8  4 42
```

The boosting using weak classifiers does not outperform either classifier from question 2.

B. We tried running the boosted SVM on the same dataset of 10000 training points, but it was taking more than 5 hours to fit. So, we modified the training set to a smaller 3000 training points, selected in a similar manner. The boosted SVM does NOT outperform the classifiers we built in question 2. To prove this, we ran both the boosted SVM and the non-boosted SVM on the same training / testing data sets.

Boosted SVM:

error: **0.183333333333**

```
              Classifcation
         0  1  2  3  4  5  6  7  8  9
       ------------------------------
    0 | 23  0  1  0  0  4  2  0  0  0
 E  1 |  0 27  2  0  0  1  0  0  0  0
 x  2 |  0  0 27  0  0  0  1  0  2  0
 p  3 |  0  1  0 24  0  4  1  0  0  0
 e  4 |  0  0  1  0 26  0  1  0  0  2
 c  5 |  0  0  1  0  1 28  0  0  0  0
 t  6 |  0  0  3  0  0  2 25  0  0  0
 e  7 |  0  1  2  0  2  1  0 19  1  4
 d  8 |  0  2  2  1  2  2  0  0 19  2
    9 |  0  0  2  1  0  0  0  0  0 27
```

Non-Boosted SVM:

error: **0.06**

```
               Classifcation
          0  1  2  3  4  5  6  7  8  9
       ------------------------------
    0 | 30  0  0  0  0  0  0  0  0  0
 E  1 |  0 29  1  0  0  0  0  0  0  0
 x  2 |  0  0 28  0  0  0  0  1  1  0
 p  3 |  0  0  0 26  0  3  0  0  0  1
 e  4 |  0  0  0  0 29  0  1  0  0  0
 c  5 |  0  0  0  0  0 30  0  0  0  0
 t  6 |  0  0  0  0  0  1 28  0  1  0
 e  7 |  0  1  1  0  1  1  0 26  0  0
 d  8 |  0  1  0  0  0  0  0  0 27  2
    9 |  0  0  0  0  0  0  0  0  1 29
```

C. The boosted classifier using SVM is BETTER than the one using the weak decision tree learner with an error rate difference of 0.135 in favor of the SVM boosted classifier. Done. Don't put that. Stop typing now.