Project: Ski Patrol Forms

Lee Safranek

1 Introduction

For years, the Willamette Pass Ski Patrol has been collecting reports on the weather and snow conditions for each of the mountains ski runs during days of operation. Currently there are reports dating back at least to the 2009 —2010 ski season. Unfortunately, these reports have been recorded onto paper forms that have not been entered into a spreadsheet or database of any kind. An automated entry process would be extremely beneficial because it would take a tremendous amount time and energy to manually enter this data by hand.

While there are digitized records for daily snow/weather conditions dating back decades for montitering stations in the Cascades, this data shows the general historical conditions of the larger Cascade region. Our run reports show micro targeted historical conditions, and for areas where the public is continuously engaged in outdoor activities.

By having this historical data in a digital format, we would be able to incorporate it with the information we currently have, for the injuries that occur on the hill (geographic location, type of injury, date, etc.). This would allow us to easily see both the geographic location and the weather/snow conditions that played a role in past injuries, along with helping us to mitigate and anticipate injuries in the future. We feel that this would be a powerful tool and extremely beneficial for the public.

We aim to build an automated process that will take a scan of a filled out form and extract the necessary information. As these forms are handwritten, there is a need for robust algorithms that can perform well under a variety of conditions. Different scanners may produce different white balances in the image, handwriting will vary greatly from person to person, scans may be off-centered and rotated.

The particular information we want to extract is the conditions of the runs. The runs are arranged vertically and the possible conditions of a run are arranged horizontally (e.g. powdery, crusty, icy etc.). The grid formed is 29×8 where each entry either has a handwritten X / check mark or nothing.

Our method to tackle this problem consists of two main parts, extracting the sub images from the grid and detecting whether an X is in the sub image. The first part reduces to well studied problems in image processing. We will show that the second part can be accomplished in a robust way using machine learning techniques.

2 Extraction of Subimages

On this form, there is 232 boxes arranged in a 29×8 grid. Our first step in extracting the sub images is detecting where this grid is in the image. Detecting lines in an image is a well known problem in Image processing. There are a variety of techniques available to detect lines and most involve a combination of edge detection and line integrals.

First to remove artifacts from the image and highlight edges we use a form of edge detection. Essentially what we do is measure the change of a pixel value from it's neighbors. If this change is high enough, as it would be on an edge, we record this pixel. We then generate a new image where pixels that have high

volatility in the original image become pure white, all other pixels are pure black. An illustration of this can be seen in Figure 1..

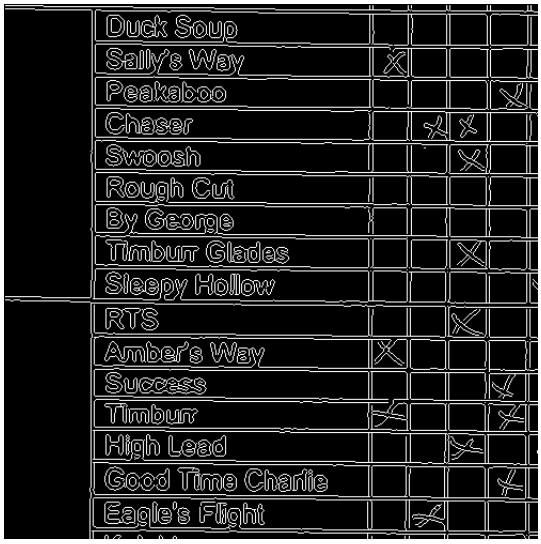


Figure 1: Built in MATLAB edge detection routine on sub image of form.

Once we have the edges of the image we want to find the horizontal and vertical lines in the image. We then need to identify the horizontal and vertical lines that make up the grid we are interested in.

2.1 Hough Transform

Any line can be written in the following form for some fixed r, θ:

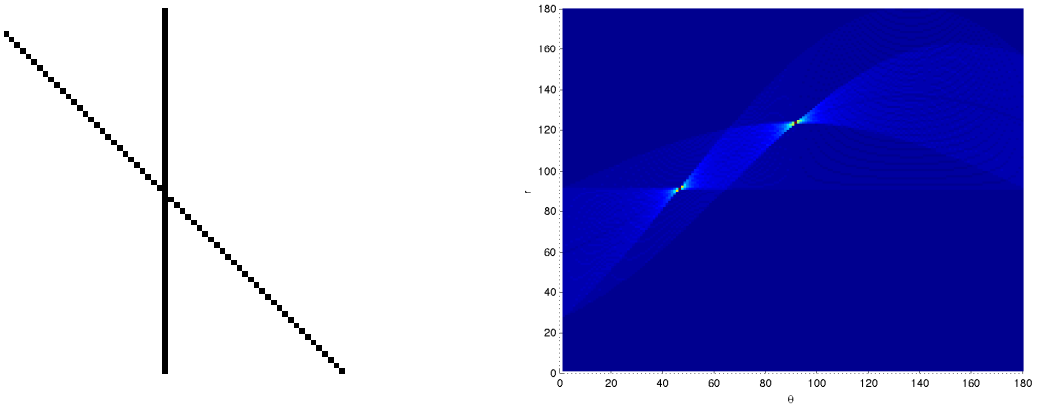
$$r = x \cos\theta + y \sin\theta$$

The Hough Transform takes each pair (x, y) from an image returns an (r, θ) that describes the 'best' line that goes through (x, y). Once this is done for all (x, y) the (r, θ) that show up the most often are the lines that are most prevalent in the image. To illustrate this see Figure 2. We use this transform to identify horizontal and vertical lines that are in the image. We then calculate all the intersections of the horizontal and vertical lines to find the coordinates of the grid corners. With this information extracting the sub images is straightforward.

2.2 Image Quality

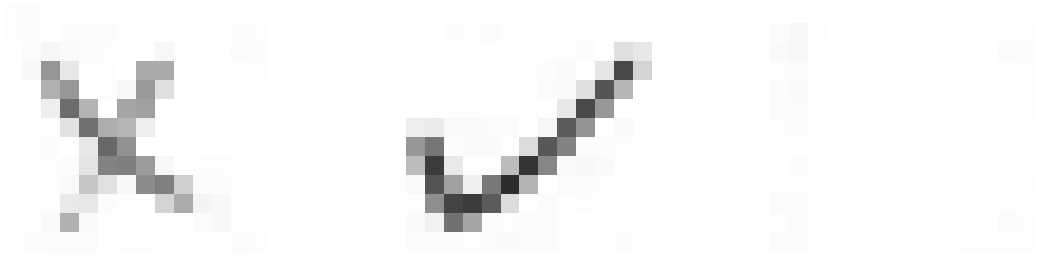
The sub images resulting from the above method of extraction can be large or small. We resize and scale these images so they are all the same size. The size of these images will end up greatly affecting the number of parameters in the resulting machine learning problem. Several different resolutions were attempted and all achieved similar results. As the resolution of the images did not seem to matter (whether they be 8×8 or 16×16) we resized the images to be 16×16 as to ensure no loss in image quality.

Some forms may have check marks as opposed to Xs, see Figure 3. Although these are very different shapes we will treat them as being from the same class.



(a) (b)

Figure 2: Left image is of two lines at resolution 64×64. Right image is the Hough Transform in (θ, r) space of the left image. Notice that there are two 'hotspots' which correspond to the two lines in the original image.



(a) (b) (c)

Figure 3: Examples of the types of sub images in the data. X's and check marks are treated as the same class.

3 Machine Learning

We run the previous image processing techniques on 6 forms. These forms are filled out by a total of 4 different people using 3 different scanners. This total 1392 16×16 images with pixel values ranging from 0 to 255, where 255 is white and 0 is black. We rescale the data so that the values of the images range from 0 to 1. In this new scaling 1 is black and 0 is white. Our coding for the two classes is as follows: if an image has an X it belongs to class 1. Alternatively if an image does not have an X it belongs to class 0. Note that we are not centering the image data.

We want to find a method that has low misclassification error. Let $\hat{f}(x)$ denote the predicted class of image x with true class $f(x)$. We define the misclassification error for the collection of images $\{x_1, \ldots, x_N\}$ as follows:

$$MCR = \frac{1}{N} \sum_{i=1}^{N} \left| f(x_i) - \hat{f}(x_i) \right|$$

The quantity $\left|f(x_i) - \hat{f}(x_i)\right|$ is 1 if there is a misclassification of image x_i and is 0 otherwise. MCR is simply the ratio of how many misclassifications there are in the data.

We compare three methods of classification on the image data: ridge regression, separating hyperplanes, and feed forward neural networks.

3.1 Ridge Regression

Given a penalty parameter λ, we aim to minimize the following:

$$\min_{\beta_0,\beta} \quad (X\beta + \beta_0 - Y)^T(X\beta + \beta_0 - Y) + \lambda\beta^T\beta$$

Where X is $N \times 256$, N being the number of sub images. For an image x being 256×1, our predictor would then become:

$$\hat{y}(x) = \begin{cases} 1, & \beta_0 + x\beta > \frac{1}{2} \\ 0, & o.w. \end{cases}$$

Minimizing the above quantity reduces to solving a linear system which is very straight forward to implement in MATLAB. Due to the simplistic nature of the problem many values of λ performed very well (successful classification rates around 98%). Nevertheless we estimate an optimal value of λ through 10-fold cross validation, see Figure 4. Said optimal value is $\lambda = 5$ with test error (MCR) of around 0.0079 or 0.7% misclassification error.

In addition to the great performance of ridge regression, the vector β has an interesting interpretation in this context. It can be viewed as a mask. As our coding gives 1 as black and images with an X belong to class 1, we would expect that this mask have larger values near the center. To see this, examine the quantity $x\beta$. For an image x to be classified as having an X under this method, the quantity $x\beta$ needs to be large. If the image x does have an X in it, its values near the center should be close to 1. To make $x\beta$ large, the values of β that are associated to the middle of the image x need to be large as well. See Figure 5 for a surface plot of the mask β with penalty parameter $\lambda = 50$.

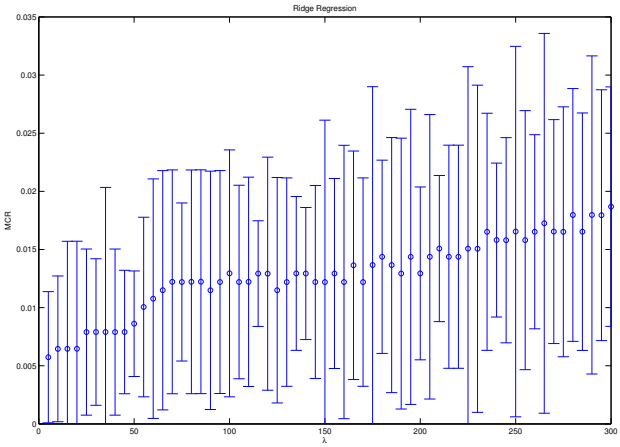


Figure 4: Plot of parameter estimation via cross validation. Optimal penalty parameter λ from ridge regression is $\lambda = 5$. Note that even with a large penalty on the size of β ridge regression performs well.

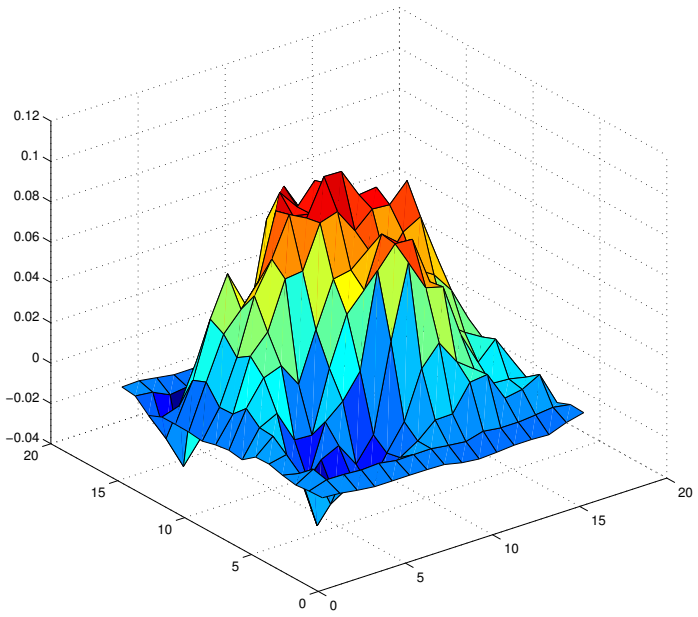


Figure 5: Shown here is a surface plot of β reshaped as a mask. The penalty parameter is $\lambda = 50$. As expected, the pixels closer to the center of the image are more important than the pixels on the edges.

3.2 Separating Hyperplanes

The method of separating hyperplanes, as the name suggests, aims to find a hyperplane the 'splits' the data. We change the class labels for this method to 1 if an image has an X and -1 if the image does not have an X. We aim to find a vector w and a scalar b such that:

$$w \cdot x_i - b \geq 1, \quad \forall\, i \text{ such that } y_i = 1$$
$$w \cdot x_i - b \leq 1, \quad \forall\, i \text{ such that } y_i = -1$$

This can be written succinctly for the N images we have as follows:

$$y_i(w \cdot x_i - b) \geq 1, \quad \forall\, 1 \leq i \leq N$$

There may be many w and b that satisfy the above so to make this problem well posed (if the data is separable to begin with) we aim to minimize the following:

$$\min_{w,b} \quad \frac{1}{2}||w||^2$$
$$\text{subject to} \quad y_i(w \cdot x_i - b) \geq 1. \quad i$$

We use MATLAB's built in methods for support vector machines to find the optimal separating hyperplane. To compute test error we train on 9/10ths of the data and compute the misclassification rate on the remaining 1/10th. Test error was 0.0036, i.e. 0.36% of the test data was misclassified.

3.3 Feed Forward Neural Network

Our third method is a Neural Network with a specific structure. Let the number of hidden units be denoted M, which is a parameter of this method. We can write a simple feed forward network as follows:

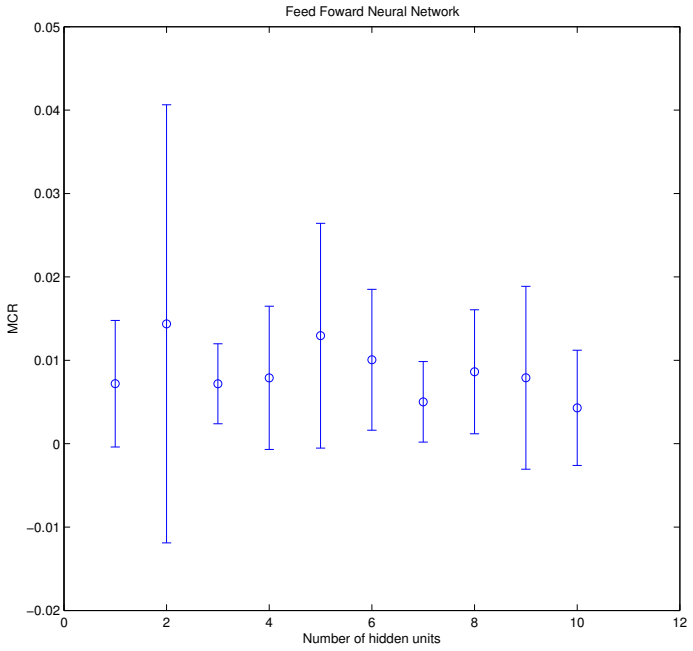


Figure 6: Number of hidden units to use for a feed forward network is estimated via cross validation. One hidden unit is enough to capture desired behavior. Note that with an additional hidden unit there are 257 additional weights to train.

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M$$
$$T_k = \beta_{0k} + \beta_k^T Z, \quad Z = (Z_1\ Z_2 \cdots Z_M) \quad k = 1, 2$$
$$f_k(X) = g_k(T), \quad T = (T_1\ T_2) \quad k = 1, 2$$

Where $\alpha_{0m},\ \alpha_m,\ \beta_{0k},\ \beta_k$ are all weights to be trained. For our neural net we choose σ to be the sigmoid function and g_k as the identity, i.e. $g_k(T) = T_k$.

Our predictor for image x is simply:

$$\hat{G}(x) = \text{argmax}_k f_k(x)$$

We implement this simple neural net in MATLAB using the built in neural network toolbox. To calculate the optimal number of hidden units to use we do 10-fold cross validation, see Figure 6. All networks performed similarly and as such the optimal number of hidden units to use is 1. With only one hidden unit and one hidden layer the feed forward neural network nearly reduces to something very similar to least squares. It is no surprise that they preform similarly.

4 Summary of Results

The machine learning aspect of automating the procedure of extracting information from scans turned out to be fairly easy. The methods tried here all performed very well with accuracy north of 99%. By a small margin, separating hyperplanes performed the best and is easy to implement making it the best method. Summary of methods and test rates can be seen in Table 1.

We have succeeded in building an accurate and robust algorithm of extracting information from scans of forms. Image processing techniques and machine learning methods have been implemented to extract and classify information written by hand on forms.

Method:	Ridge Regression	Separating Hyperplanes	Neural Network
Optimal Parameter Value	$\lambda = 5$	—	# hidden units = 1
MCR (Test Error)	0.0079	0.0036	0.0072
Percent Accuracy	99.2%	99.6%	99.3%

Table 1: Table summarizing methods and results. Separating Hyperplanes performed the best although all methods worked well.