# CSCI 181 / E-181 Spring 2014 Practical 3

Kaggle Team "Capt. Jinglehiemer"

David Wihl
davidwihl@gmail.com

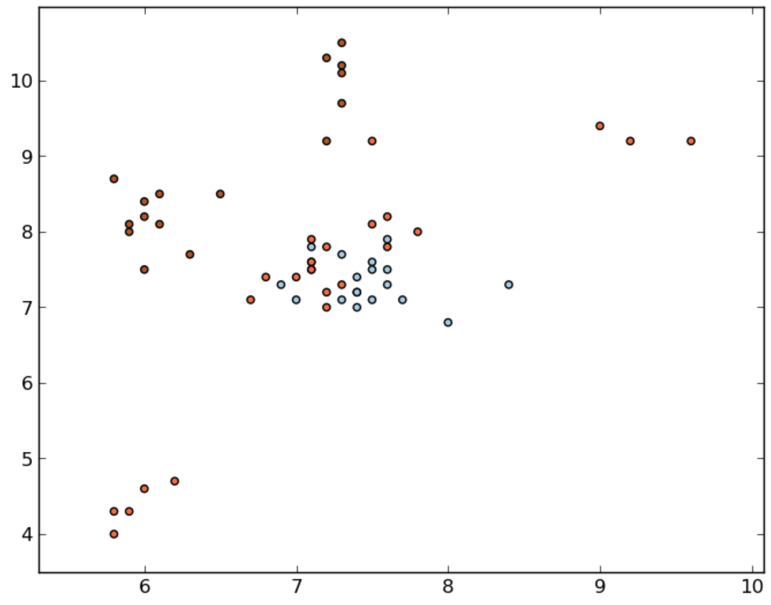
Zack Hendlin
zgh@mit.edu

March 13, 2014

## Warm-Up

We consider two approaches for classifying fruits (with length and width measurements provided) into one of three categories.

It is important to note that the data here is not perfectly linearly seperable, as can be seen in the plot below:



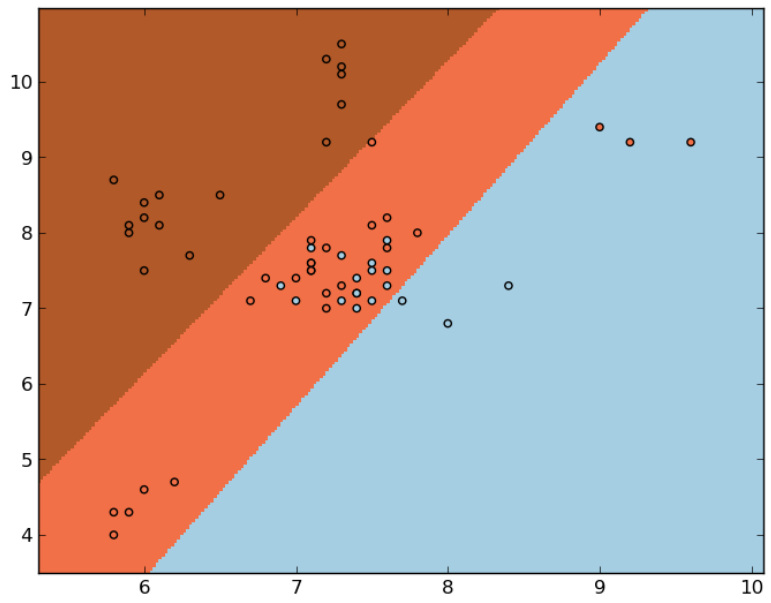
So the challenge becomes how we can best define mutually exclusive regions of the graph.

## Logistic Classification

Multiclass logistic classification seeks to define the weights that minimize:
$E(w_1, w_2, ..., w_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} = t_{nk} ln_{nk}$ , as given in equation 4.108 in Bishop.
Here our $w$ vector will have be $kx3$ as we have a variable for height, weight, and then an offset term $w_0$.
Since $k = 3$, we fit 9 weights so as to minimize the error function. We use the Broyden Fletcher Goldfarb Shanno algorithm (BFGS) as implemented in the Scipy package. We select this because it provides a better approximation to the Hessian at each step.
The error function achieves its minimum at 34.769598, and takes 44 iterations to converge. Once we have the weights:

$$w = \begin{bmatrix} -7.35269067 & 2.48983592 & -1.17947522 \\ -3.65199071 & 1.66456348 & -0.81582258 \\ 14.00478476 & -4.15564072 & 1.99410538 \end{bmatrix}$$

we then use in the softmax function: $y_k(\phi) = exp(a_k)/\sum_{j=1}^{K} exp(a_j)$
where $a$ is simply the dot product of $x$ and $w$,to evaluate which class has the highest likelihood for a particular data point. The point is then assigned to the class which has the highest class-conditional density.
We get the resulting classification.



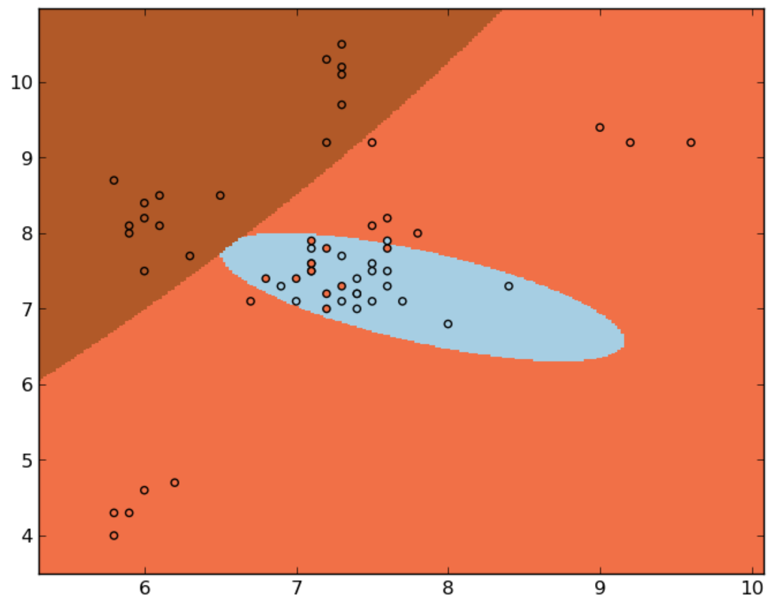
## Generative Model Classification

Generative models attempt to determine, for each class $k$, a likelihood that a data point is generated by that particular class.
We first calculate the prior likelihoods for each class $P(c_k)$
And then recognizing we want $p(C_k|x_n)$ by applying Bayes rule, we need: $p(x_n|C_k)$.
The multivariate normal is given by:
$$f_{\mathbf{x}}(x_1,\ldots,x_k) = \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$
where k=2 in our case because we have (1) height and (2) width.
This gives us a probability density function for each class. We then apply Bayes rule:

$$p(C_k|x_n) = p(x_n|C_k)p(C_k) / \sum_{j=1}^{K} p(x_n|c_j) * p(c_j)$$

and note that we can ignore the denominator since it is the same for all classes.

We then have an assignment for each point to the highest likelihood class as shown in plot below.



### Warmup Summary

As we can see from the graphs in the previos sections, allowing a non-linear decision boundry (as we did when we used the generative classifier) allowed more flexibility to capture clusters of points and indeed was more accurate at capturing the variation observed in the data.

## Classifying Malicious Software

### Preliminary Data Analysis

This problem was significantly more complex than the previous practicals due to a number of factors: unbalanced distribution of the targets, sparsity of the data, conversion of hierarchical XML data into rectangular matrices, and unclear value of certain attributes in the data.

Significant experimentation was required to iterate over feature engineering, classifier choice, permutations of ensembles of classifiers and hyper parameter optimization.

The first step was a significant refactoring of the sample code in order to separate vectorization, classification, validation and submission.

## Using Cross-validation

Five-fold cross validation was used on the training data, with runs with train / CV data pairings of 70/30, 80/20 and 90/10 for each classifier.

Very early we created a test harness that enabled testing of $n$ iterations (default 5) over cross-validation sets consisting of 30%, 20% and 10% of the data. This proved very valuable as we experimented. In total, we ran over 200 cross-validation tests each consisting of 15 different passes.

## Per File Feature Engineering

The first vectorizer took simple counts of each different type of system call as well as per-process (e.g. 'startreason', 'terminationreason', 'username', 'executionstatus', 'applicationtype') and per-file metrics coupled with an unoptimized logistic regression. This enabled us to achieve a 75% accuracy on the Kaggle submission within just a day or two.

## Comparing Classifiers and Tuning Hyperparameters

We then attempted a number of different classifiers (SVM, kNN, Gradient Boosting). Results are summarized in TODO.

By making multiple cross-validation runs, we tuned each classifiers hyper parameters.

## Pruning the Dimensions

We took two complementary approaches to prune the dimensions. 1. We examined the weights of each of the classifiers to see which dimensions were not producing a signal. Using the mean and standard deviation of the resulting matrix, we noted any features where abs(mean) $< 0.001$ and std $<0.01$. 2. We exported our Design Matrix to a CSV and then used tools in R to run a linear regression and see the predicted values of the coefficients.

We then pared back approximately 20 features that provided no value (and ran cross validation again to recheck of course).

## Combining Classifiers

We examined and experimented with several options for combining the classifiers (AdaBoost, Random Forests). We wanted to attempt a novel approach to combining classifiers (using predict_proba to find what level is >98% is appropriate for a given classifier. Then if the classifier, such kNN, had a high confidence level, it was given the vote. If one classifier had a significantly higher confidence level (determined empirically to be 0.4), it was given the vote. If no classifier had a high confidence, it was given to Logistic Regression as it generally had the lowest error rate.

We actually found that combining two different sets of classifers worked under different situations. For instance, we found that the summary features had the most overall explanatory power (XX percent), but that more specific information on each process was useful in specifically identifying certain cases.
Interestingly, we found that the hyper-detailed information on each process feature was actually less useful on average than more aggregated features. This points to the importance of feature engineering and trying to capture key elements of the underlying data rather than simply adding features without as much thought to why they might make sense and if they are the best representation of key attributes.
While we initially had them as seperate models, we instead developed a 'meta classifier' which we trained to simply discriminate which of the two models was likely to perform well on that type of potential malware. We used this model to then select between the two model prediction

## Feature Engineering

NOTE Aggregate Features per training file: selected all process features (e.g. 'startreason', 'terminationreason', 'username', 'executionstatus', 'applicationtype') and summary thread features (num of each type of system call).

We created second, separate model of per system-call metrics, consisting of items such as 'successful' attributes. Attributes of type string that had potential value were hashed into a numeric form. This resulted in a design matrix of over 3000000 rows $\times$ 14 features.

We then applied Logistic Regression to this new design matrix and rechecked, as always with multiple cross-validation passes and again pared back any features that did not have a significant signal.

## Quality Check prior to Submission

After a couple of poor submissions due to bugs in our algorithms, we implemented a sanity checked for submissions. This would useful in real world scenarios to ensure that the predictions are within a reasonable tolerance prior to putting new models into production. The sanity check ensured that all test data had a predictions within the possible values, and then displayed a distribution of distribution to ensure that no unusual skews were occurring.

## Exotic Classifiers

We examined and did preliminary experiments with two different approaches: Exemplar SVM and Theano / Deep Learning. While the reading was interesting, in both cases, the quality of the code and library precluded us from pursuing it further in the time available. We hit a bug in Theano when using the Scipy CSR array. We intend to continue exploring Theano outside the scope of this exercise in the future.

## Conclusion

We conclude from this practical that considering what classifiers work well over what ranges is extremely important. That is, when we combined models to allow them to effectively apply over different 'ranges' of the data, we were able to capture more nuanced effects than if we had simply attempted a single set of classifers that was applied to all data across the range.

This idea – which is also true in decision trees and random forests – effectively allows the function mapping the data to be more specific to the variability observed in the data and points to an important idea for us to consider in future machine learning projects.

This was a valuable exercise in understanding the state of the art in multi-class classification with real-world, sparse hierarchical data. Clearly, *there is no free lunch.*[1]

A more general approach to attacking this type of problem requires more research and investigation. The permutations of features, classifiers and hyper parameters grow geometrically without many significant tools to provide automated means of attacking the problem. This is an area we would be interested in pursuing further research.

[1] Wolpert, Macready, *No Free Lunch Theorems for Optimization* http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

We were surprised how little guidance there was in the literature for this type of problem. We scanned through several books, journals, articles and web sites, but found little external guidance that was applicable in this set of circumstances. Clearly many people have attempted to find more general means of automated multi class classification.

## Background Reading

[1] Multiclass SVM classification http://svmlight.joachims.org/svm_multiclass.html

[2] Exemplar SVM on github (octave / matlab) https://github.com/quantombone/exemplarsvm

[3] One against all or One against All http://hal.archives-ouvertes.fr/docs/00/10/39/55/PDF/cr102875872670.pdf

[4] Classifier comparison: http://scikit-learn.org/stable/auto_examples/plot_classifier_comparison.html