

# Machine Learning Algorithms for Cancer Diagnosis

Abraham Karplus

Santa Cruz County Science Fair 2012

# Contents

<b>Overview</b>	<b>3</b>
<b>Cancer</b>	<b>4</b>
Gene Expression Levels . . . . .	4
<b>Machine Learning</b>	<b>5</b>
Testing Methods . . . . .	5
Performance Measures . . . . .	5
<b>Algorithms</b>	<b>7</b>
Majority . . . . .	7
Nearest Neighbors . . . . .	7
Decision Tree . . . . .	8
Best Z-Score . . . . .	8
Extension . . . . .	9
<b>Python</b>	<b>10</b>
<b>Breast Cancer</b>	<b>11</b>
Problem . . . . .	11
Results . . . . .	12
<b>Colorectal Cancer</b>	<b>13</b>
Problem . . . . .	13
Results . . . . .	14
<b>Conclusion</b>	<b>15</b>
<b>Future Work</b>	<b>16</b>
<b>Acknowledgements</b>	<b>17</b>
<b>External software</b>	<b>18</b>
<b>References</b>	<b>19</b>

## Overview

Machine learning algorithms are computer programs that try to predict something (behavior, cancer type, what an image is of, stock market fluctuations, etc.) based on what situations caused what results in the past. The eventual goal of machine learning in cancer diagnosis is to have a trained machine learning algorithm that, given the gene expression levels or other data from a cancer patient, can accurately predict what type and severity of cancer they have, aiding the doctor in treating it.

This project is a comparison of several different machine learning algorithms, comparing their performance on diagnosing cancer from gene expression level data. The two datasets that were used were a breast cancer dataset, classifying cancers into basal and luminal, and a colorectal cancer dataset, determining if a cancer has a mutation in the p53 gene.

This project compares four different machine learning algorithms: Decision Tree, Majority, Nearest Neighbors, and Best Z-Score (an algorithm of my own design that is a slight variant of the Naïve Bayes algorithm). Initially, I guessed that the Decision Tree would perform best simply because it is a very widespread machine learning algorithm.

# Cancer

Cancer is not a single disease, but rather many related diseases that all involve uncontrolled cellular growth and reproduction. Cancer is the leading cause of death in the developed world and second in the developing world, killing almost 8 million people a year.

Since cancer is many diseases, treating an individual cancer requires knowing what abnormal behaviors are happening inside the cells. This information can also aid in understanding the mechanisms behind cancer, which can lead to new treatments.

One very useful set of data is the cells *gene expression levels*.

## ***Gene Expression Levels***

Genes contain the information needed to make proteins (which do much of the work in a cell), and knowing what proteins a cell is making can tell a lot about how the cell is doing. However, it is difficult to directly measure the protein levels, so instead biologists measure levels of *mRNA*, the “messenger” between the genes and the *ribosomes*, which are the protein factories of a cell.

To determine what genes are being expressed, mRNA is extracted from the tissue and copied to cDNA, which is tagged with a fluorescent dye so that it can be seen. This dyed cDNA is then put on a *DNA microarray* or *gene chip*, which is a chip containing thousands of little spots of known DNA, which the cDNA binds to. By observing where on the chip there is fluorescent dye, the cDNA, and hence the gene expression levels, can be determined. One gene chip can measure expression levels for tens of thousands of genes.

The data for this experiment are the logarithms of these expression levels, since there may be several orders of magnitude difference between the expression levels of different genes.

# Machine Learning

Machine learning is the subfield of computer science that studies programs that generalize from past experience. This project looks at *classification*, where an algorithm tries to predict the label for a sample. A *sample* is a single set of *feature* data (here, gene expression levels for a cancer patient) plus a *label*, which is what category (for example, basal or luminal) the sample falls in. The machine learning algorithm takes many of these samples, called the *training set*, and builds an internal model. Using this model, it can then predict the labels of other samples, called the *testing set*.

## Testing Methods

A common problem with machine learning is *overfitting*—bias towards the samples in the training set. An algorithm may perform well on the training set, but not generalize to other cases. Therefore, *cross-fold validation* is used, where the data is partitioned into training and testing data several different ways (each partition is called a *fold*), and the performance of the algorithm is averaged over all of these folds. Which samples are used for training and which for testing can impact the measured performance, so the cross-fold validation is done many times with many different folds.

My experiments used 5-fold cross validation: the data was split into 5 equal subsets, each of which was held out in turn as the testing data while the algorithm was trained on the remaining 4 subsets.

## Performance Measures

The simplest measure of machine learning algorithms is *accuracy*, the proportion of correctly classified samples. This measure is flawed if the data is unbalanced—in a dataset that was 90% label A, an algorithm that always guessed A would have an accuracy of 90% but no utility.

Other measures tally the samples of each pairing of actual and predicted labels (called the *confusion matrix*). When there are only two possible labels, the cells of the confusion matrix are named as follows:

Breast Dataset	Actually Luminal	Actually Basal
Predicted Luminal	<i>True Negatives (TN)</i>	<i>False Negatives (FN)</i>
Predicted Basal	<i>False Positives (FP)</i>	<i>True Positives (TP)</i>

One popular measure is *Matthews Correlation Coefficient* (MCC):

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC ranges from -1 for the worst possible prediction to 1 for a perfect prediction. An algorithm that guesses randomly has  $\text{MCC} = 0$ . I used a generalization of MCC (Gorodkin 2004) that works for any number of labels, but reduces to MCC for two labels.

Another performance measure is *information gain*, based on the *encoding cost* of sets of samples. The encoding cost of a set is a function of the number of each type of label in the set.

Using an auxiliary function  $h(x) = x \log(x)$ , we define encoding cost

$$H(x, y, \dots) = h(x + y + \dots) - h(x) - h(y) - \dots$$

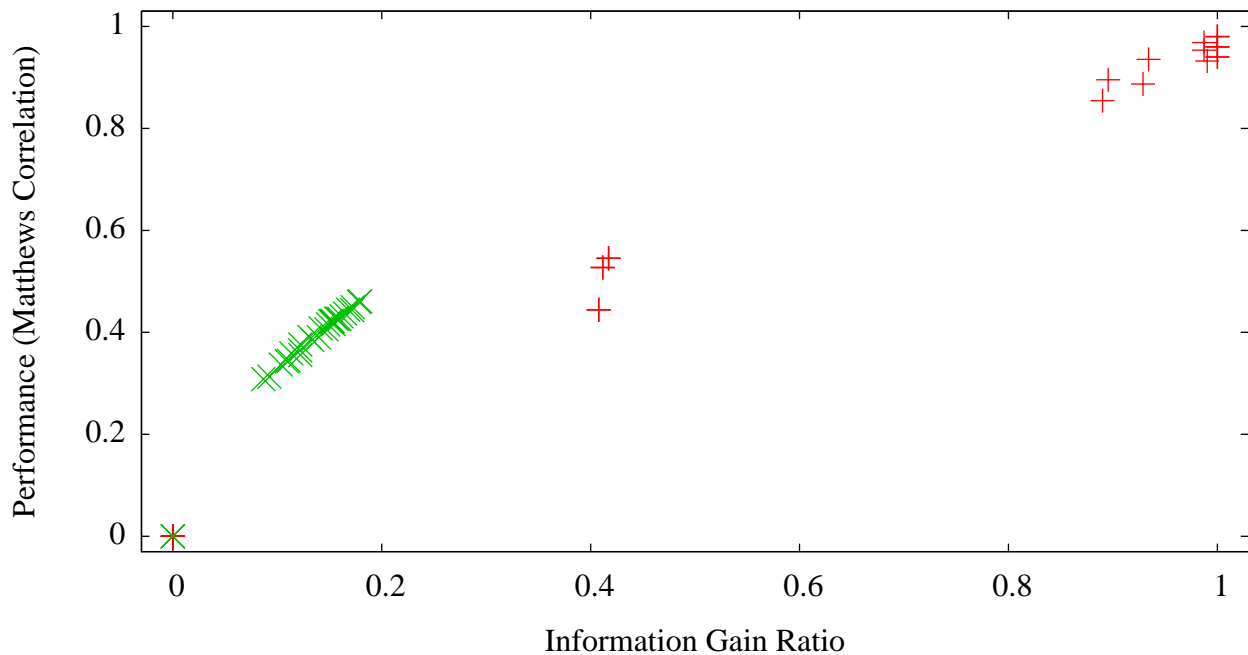
Information gain is the difference between the total encoding cost before and after making predictions:

$$\text{IG} = H(TN + FP, TP + FN) - (H(TP, FP) + H(TN, FN))$$

*Information gain ratio* is the ratio of information gain to the encoding cost  $H(TN + FP, TP + FN)$ , so ranges from 0 (no improvement) to 1 (perfect separation by label).

I compared MCC and information gain ratio. They appear to measure the same thing. MCC spreads the results farther apart, though, so I decided to use MCC in the main experiments.

Comparison of performance measures

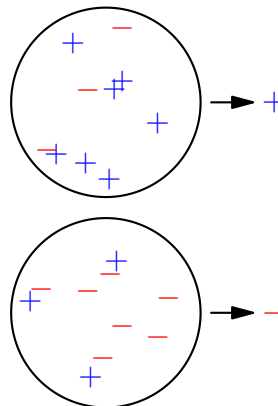


# Algorithms

In Python, I created an *abstract base class* for the general idea of a classification algorithm. Each specific algorithm is then a *subclass* of this base class, implementing the training and labeling methods.

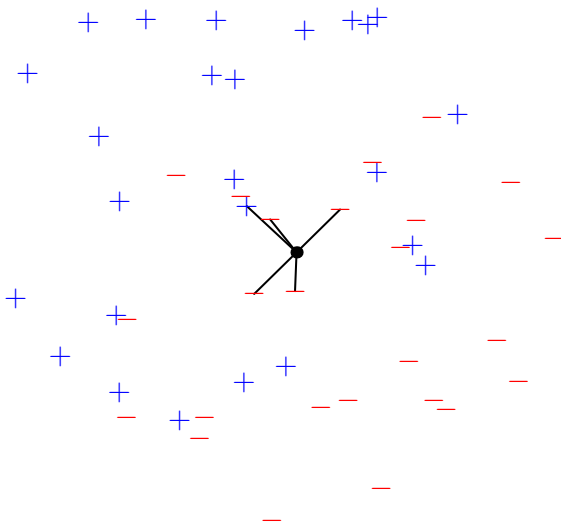
## Majority

The Majority algorithm is one of the simplest possible. To train, it merely counts the number of samples with each label. To predict, it guesses the label that got the most hits in the training data. This makes it completely useless for any actual prediction, but Majority serves well as a baseline or null model to compare other algorithms to.



## Nearest Neighbors

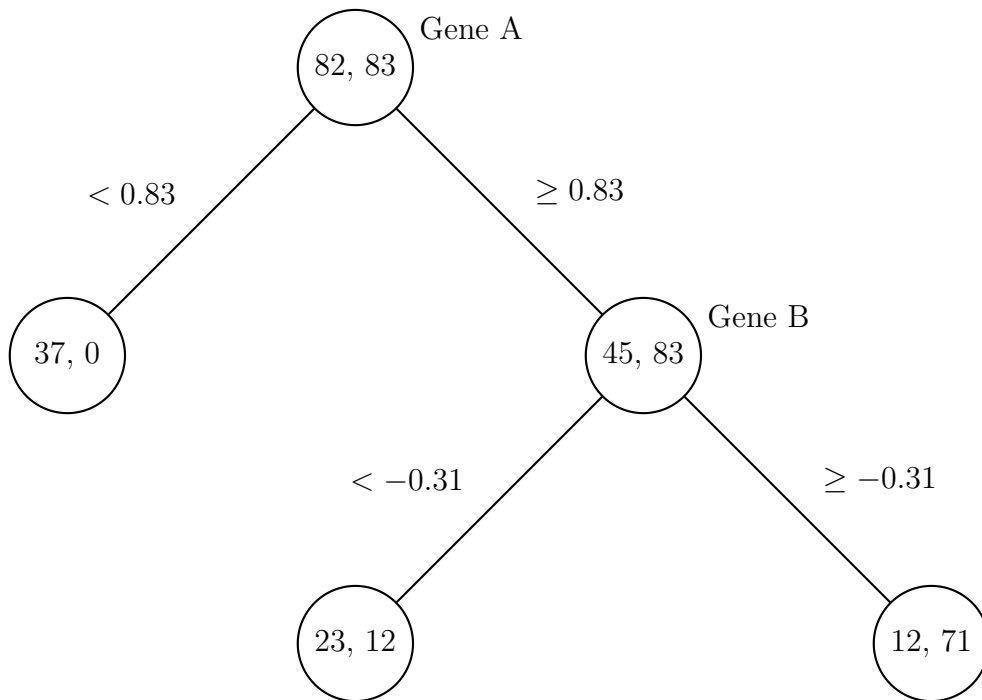
The Nearest Neighbors algorithm is based on the idea that samples with similar-valued features tend to have the same label. To predict a label, the algorithm looks at the  $k$  training samples closest to the test sample, and predicts the most frequent label of the  $k$  samples. Distance is determined by the samples position in the *feature space*, an  $n$ -dimensional space with each dimension corresponding to a feature. When  $k$  is specified greater than or equal to the number of training samples, Nearest Neighbors is identical to Majority, and so useless.



Some additional refinements are possible to the algorithm in two main ways. First, the distance metric used to figure out which samples are closest can be changed. The standard is of course Euclidean distance, but other metrics may give better results for some data sets. Second, samples can be weighted based on their distance from the test sample, so that the labels of the closest samples count for much more. I have implemented neither of these variations yet, but plan to.

## Decision Tree

The Decision Tree algorithm works by finding the specific features that divide the training data by label. It goes through all features and all possible *cutpoints* (cutpoints are values for the feature—all samples with a value for that feature below the cutpoint go to one side, while the others go to the other side) for each feature to find the one that best separates the training data by label. There are many ways to determine which split is best; the one I use is information gain. Whichever split lowers the encoding cost the most gets selected. If this procedure is only performed once, the algorithm is a *decision stump*. To be a true decision tree, it must recursively try and find a new feature and cutpoint for each of the subsets of data from the previous split, repeating until the maximum depth is reached or the training data is split perfectly by label.



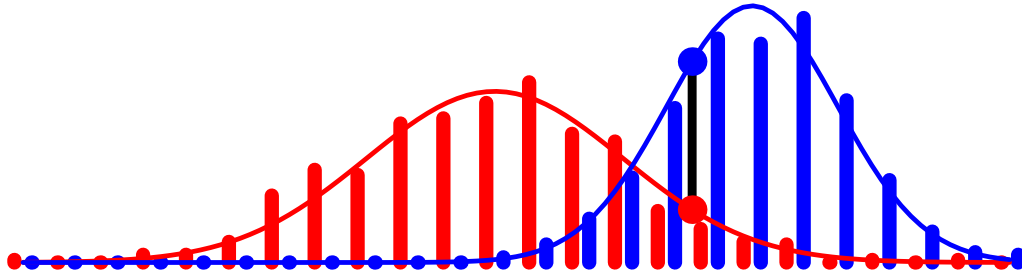
## Best Z-Score

The Best Z-Score algorithm is my own variant of the very popular Naïve Bayes algorithm. The Naïve Bayes algorithm uses Bayesian statistics for prediction, making the “naïve” assumption that each feature is independent of all others and calculating the Bayesian probability of the label given the feature. It predicts the label deemed most probable by multiplying together these conditional probabilities.

Best Z-Score makes the further assumption that the values for each feature for each label are in an approximately Gaussian distribution. Rather than calculating probabilities, it calculates the mean and standard deviation for each feature using the training samples with each label in turn. For prediction, each feature of the test sample is converted for each label to a *z-score*, which is a measure of how many standard deviations a value is from the mean.



The algorithm then predicts the label that has the lowest total squared z-score summed over all features. However, as I originally wrote the Best Z-Score algorithm, it had a bug which resulted in each squared z-score being weighted by the standard deviation for that feature and label. To fix the bug I generalized the Best Z-Score algorithm so that it took a parameter which specified what power of standard deviation to weight the z-score by. The intended algorithm corresponds to a parameter value of 0 and the buggy implementation to a value of 1. I also tested a parameter value of 2, which is equivalent to using just differences from the means and ignoring standard deviations.



### ***Extension***

It is very easy to add additional labeler algorithms to the program, and does not require modifying any existing files. Here are instructions on how to add a labeler, for a hypothetical algorithm called Foo Bar. Create a file in the main code directory called `l_foobar.py` which contains a class definition for the class `FooBar`. `FooBar` must inherit from the abstract base class `BatchLabeler` in `abstract.py` in the code directory. `FooBar` must implement the methods `learn_batch` and `label_one`, and may also implement `setup`. `learn_batch` takes two arguments: a Numpy array of feature data (each row is a sample, each column a feature), and a Numpy array of labels (one-dimensional). The method must train the algorithm on the given training data. `label_one` takes a single argument, a one-dimensional Numpy array containing the sample to be labeled. It must predict the label for the given test sample. If implemented, the method `setup` will be called with any additional arguments to be passed to the labeler on setup, such as the maximum depth for a decision tree. If needed, the labeler instance contains an attribute `known_labels` with a set of all labels seen in the training data.

# Python

Python is a popular object-oriented scripting language created by Guido van Rossum that promotes code reuse, easy object orientation, and quick development. The adjective “*Pythonic*” refers to code that is in the style and philosophy of Python, which emphasizes clarity, readability, and simplicity.

Because Python is an interpreted language, it runs slower than compiled languages. To alleviate this, I used Pyrex, a program to convert Python code into equivalent C code, for the information-gain calculation. This optimization sped up Decision Tree several-fold.

Python is dynamically typed, so uses significantly more memory than a statically typed language. To help with both memory and speed, I used the Numpy package, which provides compact and fast array data types.

# Breast Cancer

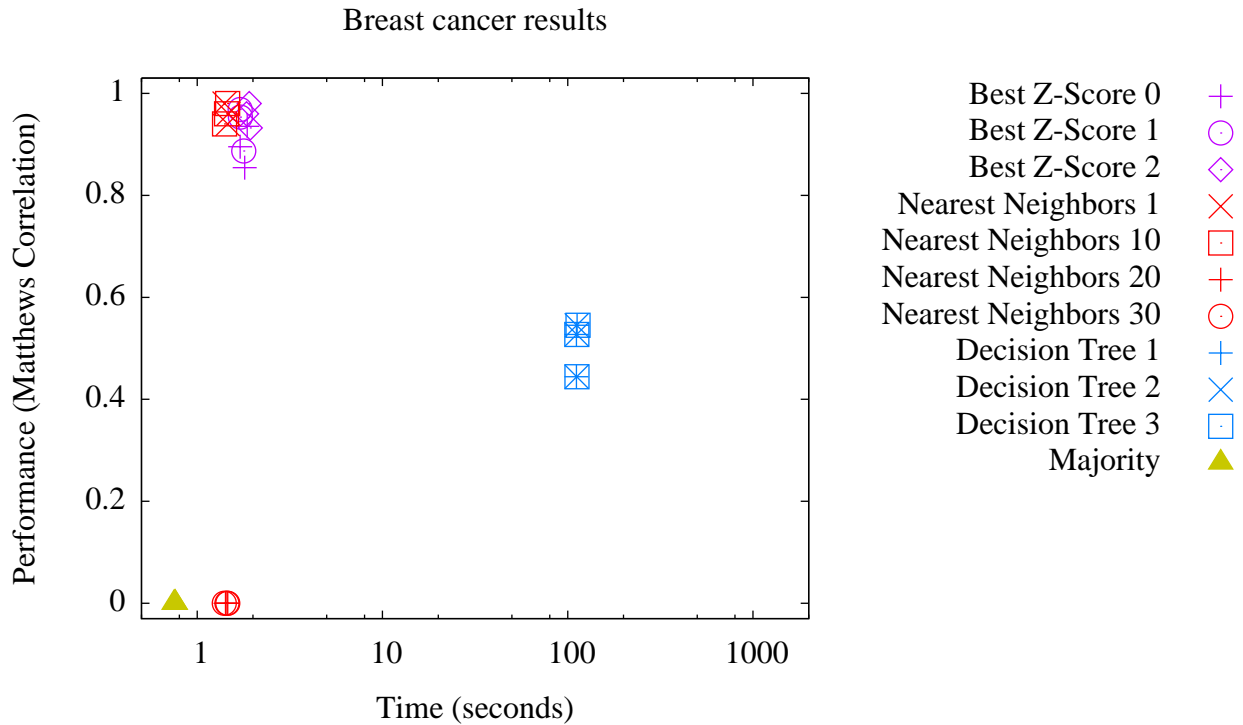
## *Problem*

Breast cancer is a very common disease, killing almost 40,000 people annually in the United States alone. It has two major subtypes, known as basal and luminal, after where in the breast they originate. Luminal breast cancer is the more common and tends to have a better prognosis and survival rate. Basal cancer tends to be more aggressive and has a higher rate of recurrence. It can be difficult, however, to distinguish between the two types. Since treatments for these two types of breast cancer differ, it is vital for doctors to be able to distinguish between them.

One way of distinguishing basal and luminal cancers is to use the gene expression levels from biopsied tumor cells. I used several machine learning algorithms to predict whether a given breast cancer was basal or luminal, based solely on its gene expression levels.

## Results

The breast cancer turned out to be a fairly easy classification problem. All three versions of the Best Z-Score algorithm did very well and were fast. Nearest Neighbors was only slightly faster than Best Z-Score. With one or ten neighbors, it performed perfectly; however, as there were only about 20 samples total in the training set, Nearest Neighbors with 20 to 30 neighbors was identical to Majority, which performed terribly. All three levels of Decision Tree performed adequately, but they took quite a bit longer than the others to train. The reason that all three levels of Decision Tree performed identically is that level 1 Decision Trees were able to separate the training data completely, and so level 2 and level 3 had nothing to separate.



# Colorectal Cancer

## *Problem*

Colorectal cancer is actually two types of very closely related cancers, colon cancer and rectal cancer. Colorectal cancer is the second leading cause of cancer deaths that affect both men and women in the United States, claiming over 50,000 lives per year. In almost half of colorectal cancers, there is a mutation in the p53 gene which results in over-expression of the p53 protein and a poorer prognosis and lower survival rate for the patient.

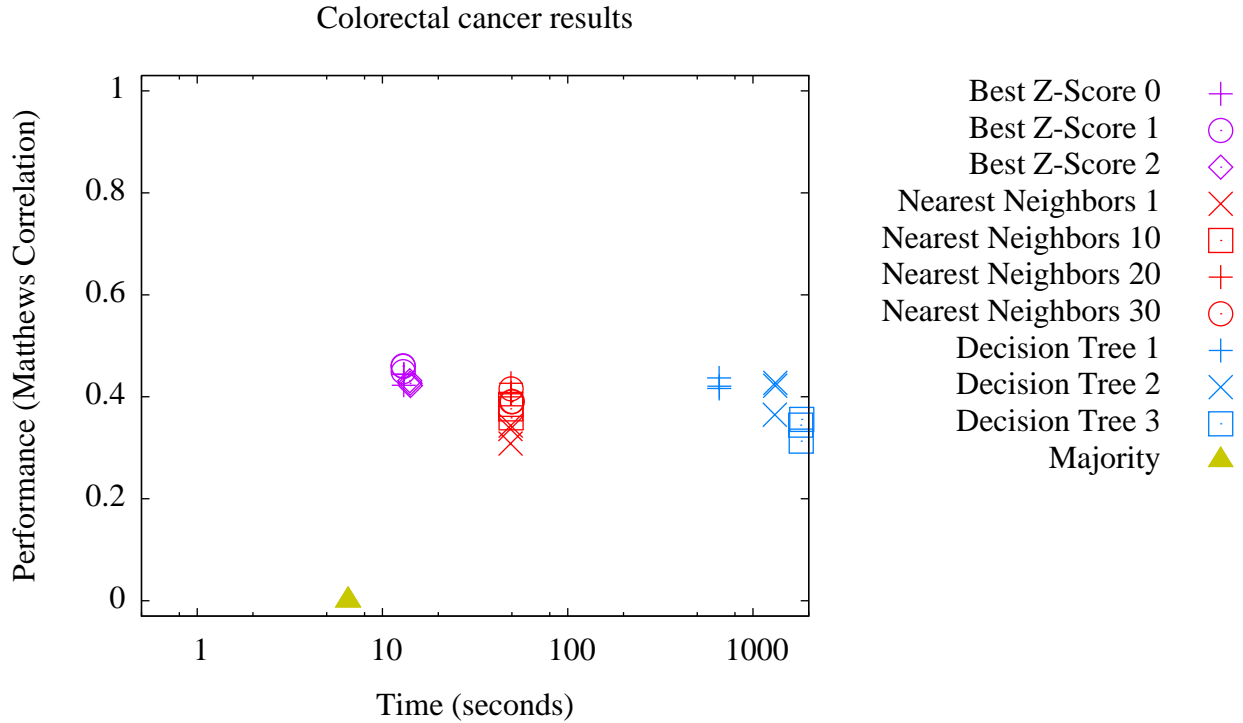
Detecting the p53 mutation is very helpful to doctors in treating colorectal cancer. Since the p53 mutation is one of the biggest genetic changes in colorectal cancer, knowledge of it is crucial to the cancer treatment.

Machine learning algorithms can predict p53 mutation from gene expression levels of tumor cells, and my goal with this project was to determine how well various algorithms could predict the p53 mutation.

The colorectal data set came from The Cancer Genome Atlas, which is a project to collect genomic, expression, and epigenetic data on cancer together with clinical information.

## Results

The algorithms performed similarly in the colorectal as in the breast cancer dataset, but with several important differences. As there were many more samples for colorectal, Nearest Neighbors never dropped to the level of Majority. Somewhat surprisingly, performance decreased for the higher level Decision Trees. Again, all three variants of Best Z-Score performed best and here took less time than Nearest Neighbors. In general, the algorithms took over an order of magnitude longer to run than they did on the breast cancer dataset. The classification for colorectal was much more difficult, with no algorithm getting a score of 0.5 or above.



## Conclusion

The algorithms that performed best (Best Z-Score and Nearest Neighbors) used all features in classifying a sample. Decision Tree used only 13 features for classifying a sample and gave mediocre results. Majority did not look at any features and did worst.

All algorithms except Decision Tree were fast to train and test. Decision Tree was slow, because it had to look at each feature in turn, calculating the information gain of every possible choice of cutpoint.

## Future Work

There are tens of thousands of features (different genes) in both the breast cancer and colorectal cancer data sets, and anything that reduced this number would speed up training decision trees. Many other algorithms also suffer from this so-called *curse of dimensionality*.

One way to reduce dimensionality is *feature selection*, where the data is preprocessed to remove features that are unlikely to be useful. Feature selection could also help many algorithms by removing noise. We could use the mean and standard deviation information from Best Z-Score to choose features with large differences in z-scores.

Feature selection is one case of *dimensionality reduction*—another method might do this by combining correlated features, rather than eliminating features. One popular dimensionality reduction technique is *principal components analysis*, which creates *components* that are linear combinations of associated features.

In addition to dimensionality reduction, I also want to implement several more algorithms: Random Forest, Support Vector Machine, Naïve Bayes, and Neural Net. For some of these algorithms, such as Random Forest, which creates many decision trees, dimensionality reduction is essential for training in a reasonable time frame.

So far, I have only tested the algorithms on two datasets, but there are many more available from TCGA and other sources, and the more datasets I test on, the more complete a picture I have of algorithm performance. Some of these datasets have features other than gene expression levels, such as copy number variation and methylation of the genome. It would be a fairly small modification for my code to handle multiple feature sets.

Eventually, I hope that the algorithms I have implemented, tested, and even developed may aid in the treatment and diagnosis of cancer in clinical situations and possibly even save lives.



## Acknowledgements

I would like to thank my father, Kevin Karplus, for the inspiration for this project as well as for connecting me with the Cancer Machine Learning Group at UCSC. In the Cancer Machine Learning Group, I would specifically like to thank Josh Stewart, Artem Sokolov, Dan Carlin, and Sam Ng for providing me with the datasets, suggesting possible algorithms and directions to take the project in, and providing feedback on my work. I would also like to thank Rina Natkin and my mother, Michele Hart, for providing support with the writing and time management for this project.

## External software

All programs were written and run with Python version 2.7, incorporating Pyrex version 0.9.9 and Numpy version 1.6.1. GNU Make 3.81 was used to automate the experiments. Data was kept in RDB format, using the RDB package version 2.9. All the plots in this poster were created with gnuplot version 4.4. The poster was prepared using Pages version 4.1. and the report with LaTeX (pdflatex version 3.1415926-1.40.11-2.2). The diagrams were made with Asymptote version 1.66 on the Art of Problem Solving website ([artofproblemsolving.com](http://artofproblemsolving.com)).

## References

- Centers for Disease Control and Prevention. *Frequently Asked Questions about Colorectal Cancer*. 2012. [http://www.cdc.gov/cancer/colorectal/basic\\_info/faq.htm](http://www.cdc.gov/cancer/colorectal/basic_info/faq.htm)
- Gorodkin, Jan. “Comparing two K-category assignments by a K-category correlation coefficient.” *Computational Biology and Chemistry* vol. 28, no. 5-6, pp. 367-374. 2004.
- Iacopetta, Barry. “TP53 Mutation in Colorectal Cancer”. *Human Mutation* vol. 21, no. 3, pp. 271-276. 2003.
- Susan G. Komen for the Cure. *Molecular Subtypes of Breast Cancer*. 2011. <http://ww5.komen.org/BreastCancer/SubtypesofBreastCancer.html>
- Wikipedia. *Cancer*. <http://en.wikipedia.org/wiki/Cancer>
- . *Matthews Correlation Coefficient*.  
[http://en.wikipedia.org/wiki/Matthews\\_correlation\\_coefficient](http://en.wikipedia.org/wiki/Matthews_correlation_coefficient)
- Witten, Ian H. and Frank, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edition. Morgan Kaufman, San Francisco. 2005.