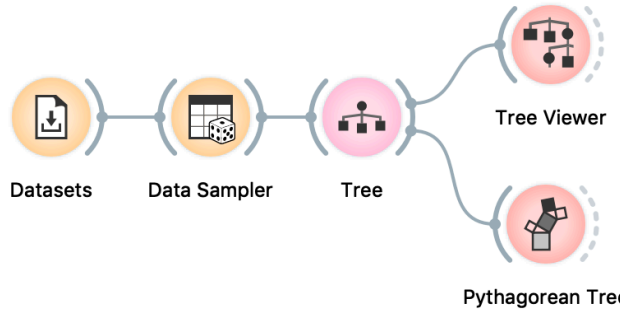
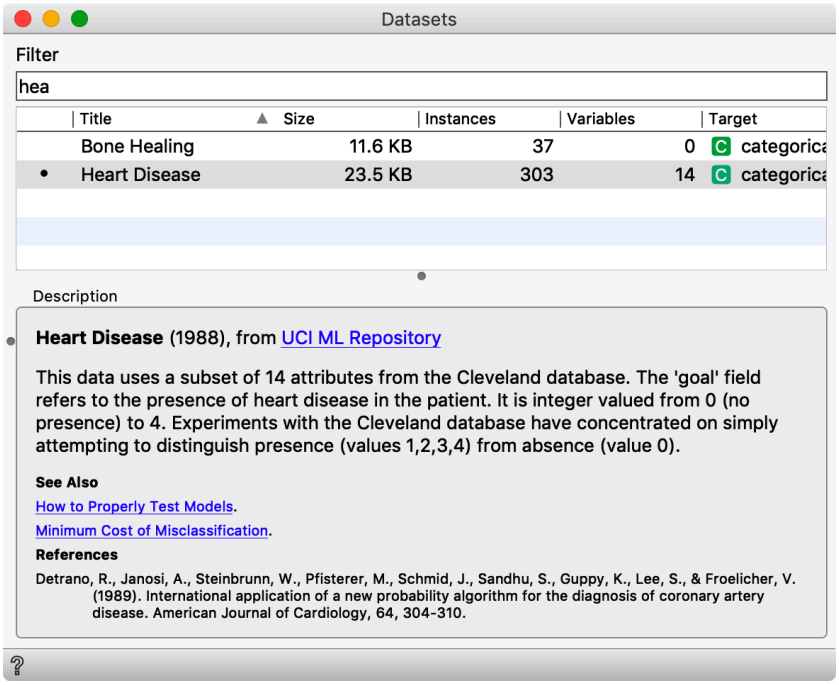
Lesson 17: Trees are not Stable

Classification trees may be sensitive to any changes in the input data. With a minor change in the data, the structure of the classification tree can drastically change. Data in biomedicine may include measurement or observation noise. Beautiful property of the trees is that they are interpretable; that is, we can "read" the model and explain it. But with the instability of the trees, if this is indeed the case, the interpretability does not make much sense.

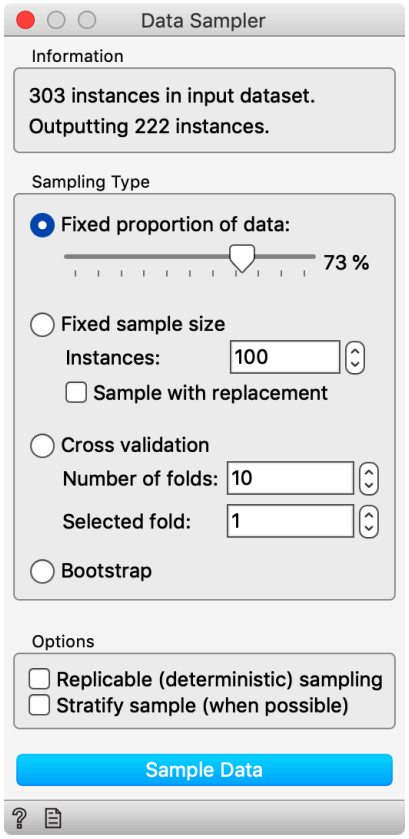
Let us observe the stability of the classification trees on an example. We will consider the data on the hearth disease with 14 attributes and a binary class variable that reports on the presence of the disease.

The trees inferred from heart disease dataset are large and do not fit well in Tree Viewer's visualization. Orange has a widget Pythagorean Tree with an alternative, more compact display.

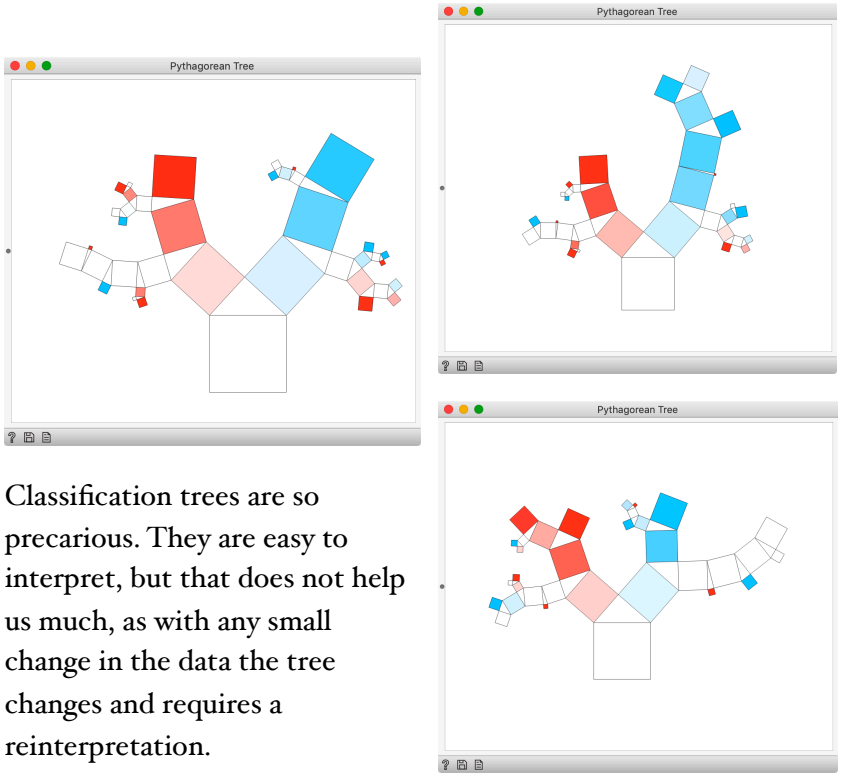




In the workflow, we sample 70% of the data, develop a classification tree, and visualize it. Keeping both Data Sampler and

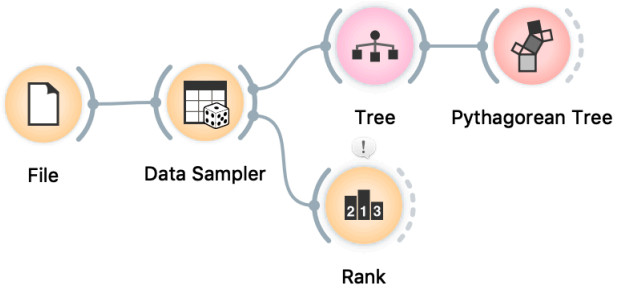


Pythagorean Tree widgets on the top, we can now create a new sample and see the changes in the tree structure. They are substantial, and it looks like that with every new sample, we obtain an entirely different tree.



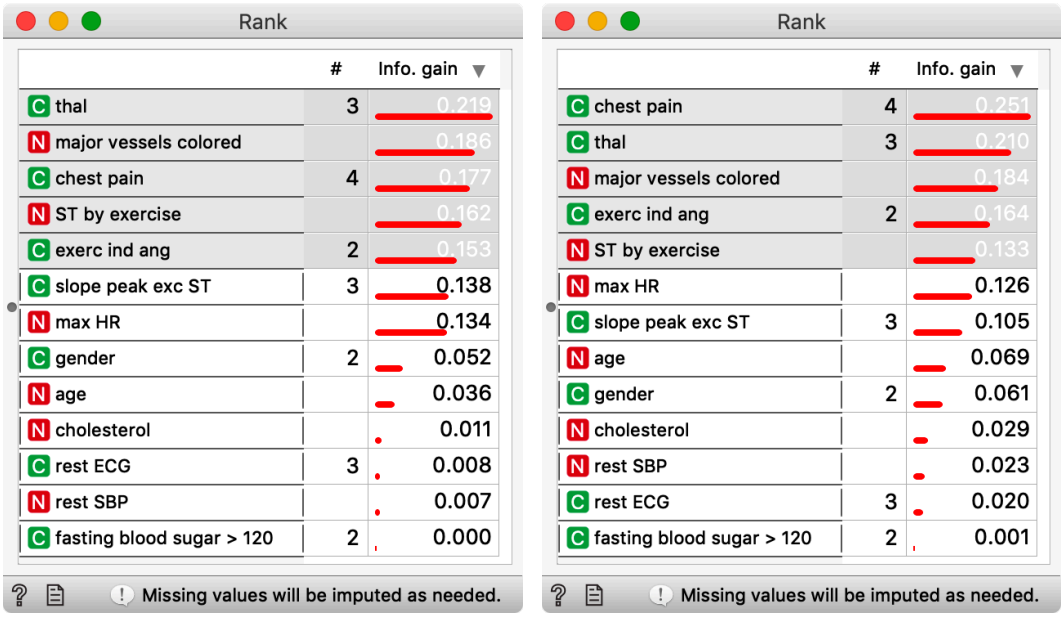
Classification trees are so precarious. They are easy to interpret, but that does not help us much, as with any small change in the data the tree changes and requires a reinterpretation.

Instability of the trees emerges as, when developing the tree, at each internal node the algorithm chooses the most informative feature for the split of the node's training data. If there are several features with comparable correlation with the class, tiny changes in the data can change the ranking of the features. Changes in the feature order on the top reflects the change in the choice of the feature that will be selected to split the data at the root.



To explore how the changes in the data impact the ranking of the features in the tree's top node, that is, for the entire data set, we can use the Rank widget. Even for the whole dataset, when the set dataset is still reasonably large, different features win for the different data sample. When the feature selected for the root of the tree changes, the structure of the entire tree would change.

The instabilities of feature ranks are even more pronounced in the lower layers of the tree, where the datasets pertinent to each node becomes smaller.



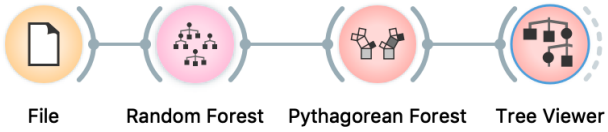
Feature ranking on entire sample of heart disease data set shows that the order of the features depends on the sample. Features most correlated with the class always appear on the top of the list, but their order changes. The tree induction algorithm would always pick the top-ranked feature for the split and the sole change in the root of the tree would result in entirely different trees.

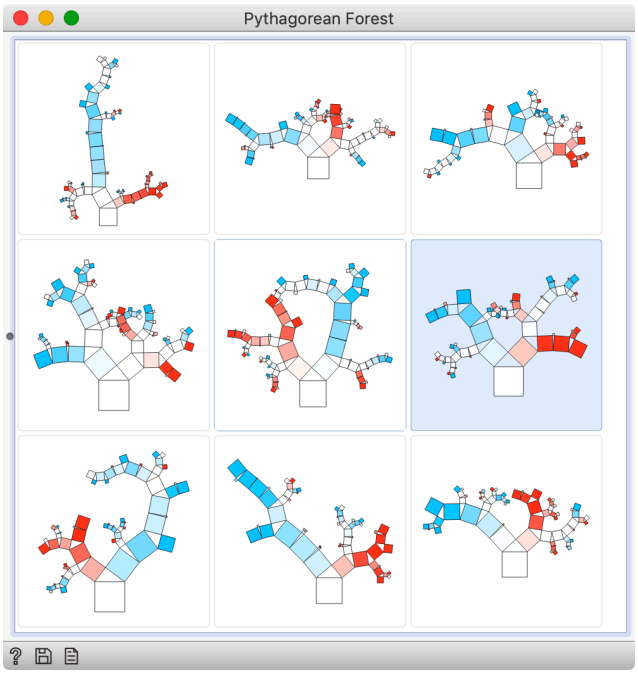
Lesson 18: Random Forests

With instability of trees, the interpretive advantage of the tree is gone. But constructing different, or better, entirely different trees from the same data set could have an advantage. Remember the TV show Who Wants to Be a Millionaire? In the show, the contestant has to answer consecutive multiple-choice questions of increasing difficulty. There is a lifeline called Ask the Audience, where the contestant asks the audience to answer the question and obtains the help by seeing the distribution of the answers. Most often, the answer that received the most votes from the audience is the right one. Such "wisdom of the crowd" approach, where the collective opinion of a group of individuals rather than that of a single expert, is today employed by social information sites. And is used in machine learning. Each of the tree, inferred from the sample of the training data, could be considered an individual. A collection of the tree would vote for the class, and a machine learning technique called random forests would then predict the class value that received the majority of votes.

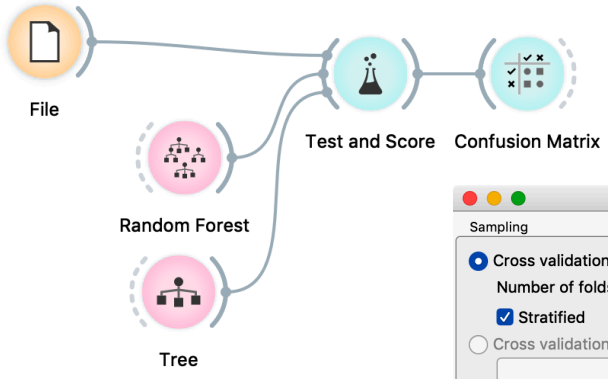
Random forest develop trees from so-called bootstrap sample, a sample of the training data that is of the same size but draws randomly from the training set with replication. To increase the diversity of the trees, the features on which the tree nodes split the data are randomly drawn from the top-ranked features.

Random forest consist of a set of trees that can be in visualized in Orange using Pythagorean Forest widget. To observe the details of the tree, a selected tree from the forest can be sent to the Tree Viewer or to Pythagorean Tree for further inspection. Visualizing the trees in the forest serves only to assure us that the inferred trees are indeed different. Else, random forest are treated as black boxes.

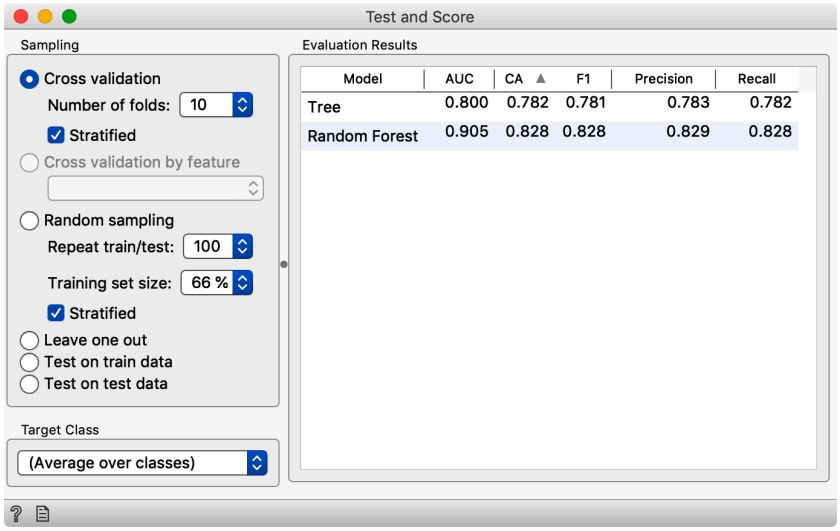




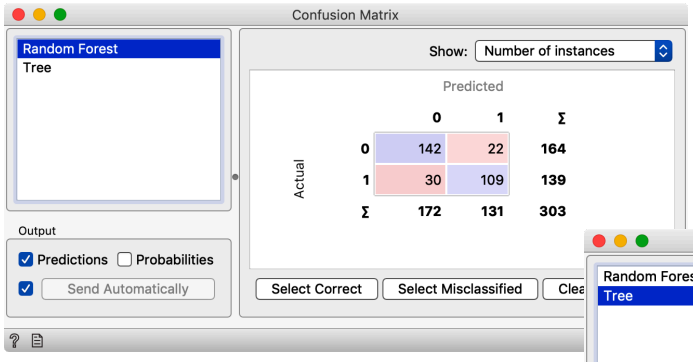
Random forests are impossible to interpret. Their only gain is in accuracy. And this could be substantial. Let us measure this through cross-validation on a heart disease data.

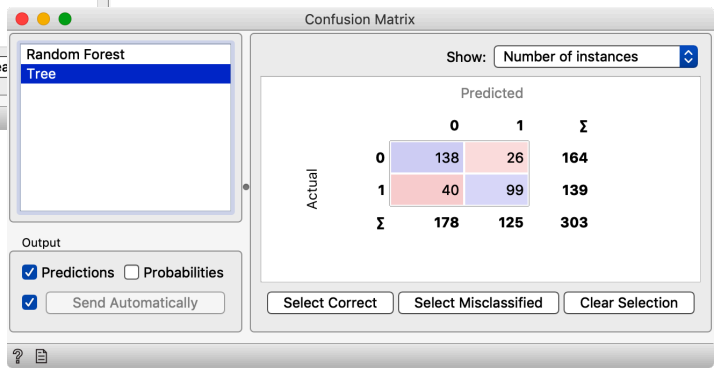


A core parameter of the random forest machine learning method is the number of trees. We have set this parameter to 100 in our experiment. Most often, a few hundred trees in the forest are sufficient for optimized accuracy, and increasing the number of trees above 500 does not yield any further gains.



The gain of the forest over the individual tree is substantial. Not just in classification accuracy (CA), but also in every other statistics that we will present in detail in one of the following lessons.

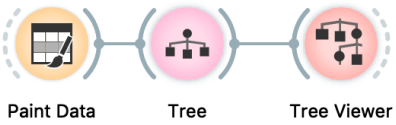


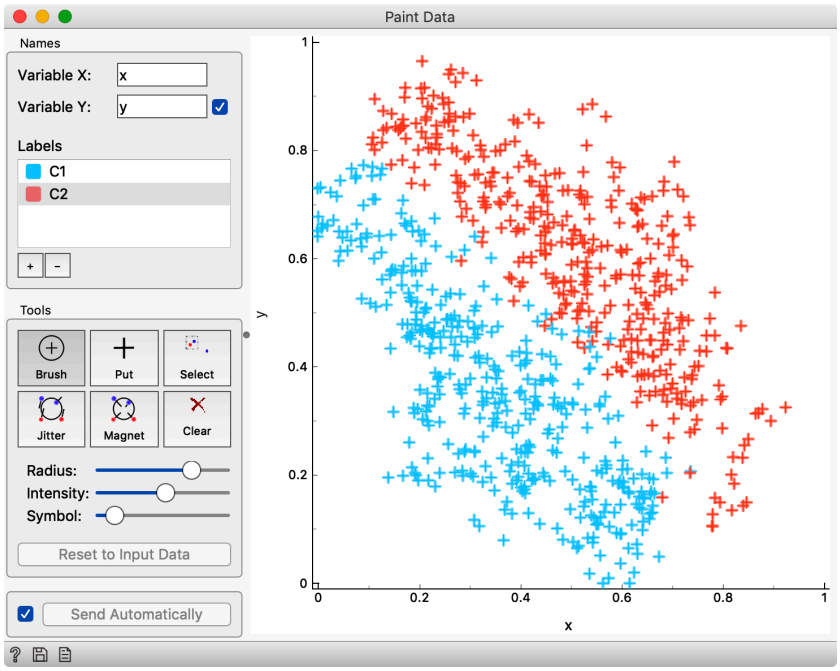


With classifications, it is always wise to check the confusion matrix. Here, we see that out 303 data instances, the trees misclassified 66, and random forests misclassified 52, with a most significant reduction of misclassification taking place for the subjects with disease.

Lesson 19: Logistic Regression

Trees "cut" the feature space according to the value of a single feature, that is, they discover "boxes" in the feature space with a prevailing class value. The method struggles with more complex shapes of decision boundaries. We can paint the data set where the classification tree would struggle. For instance, for the painted data set as depicted below, the inferred tree is quite complicated and contains 22 internal nodes and 11 laves.

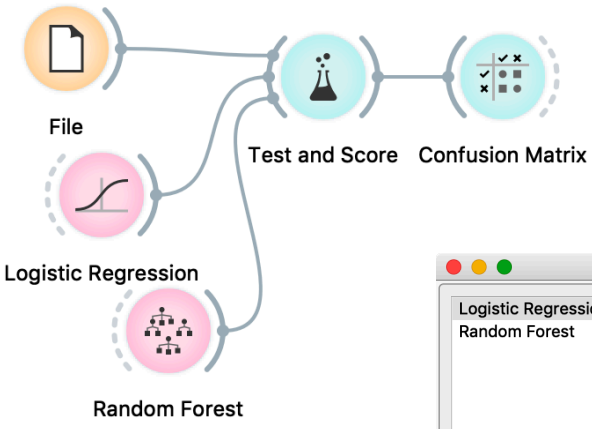


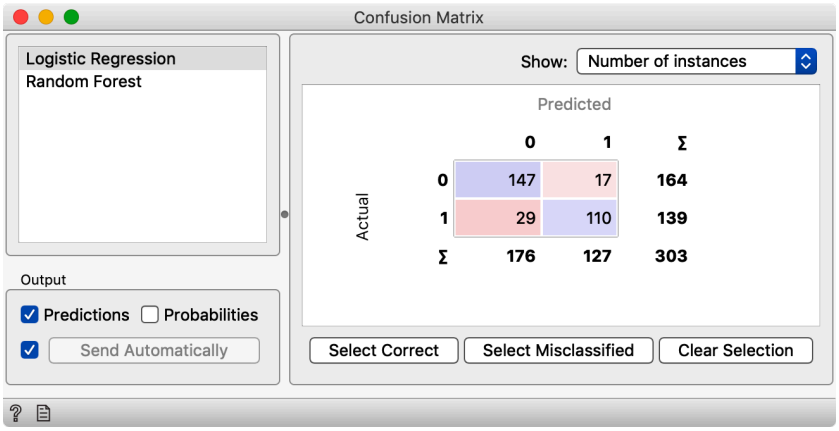


Possibly the best model for the data shown above is a line that splits blue and red points. We could also assign the class probabilities according to this separation line: the further away from the line a data point, the more likely it belongs to one or the other class. The data points on the separation line are those for which we cannot decide on a class, and where the class probabilities are equal, that is. Notice that a line could separate the two classes in two-dimensional feature space, whereas for three or higher dimensional spaces we need a plane or a hyperplane.

The model that follows our reasoning above, and from the data infers the planes that separate the two classes is called logistic regression. The "language" of this model is linear as the planes are flat. Surprisingly, though, this model behaves rather well and is often comparable to the random forest. On the heart disease data set, the logistic regression even slightly beats the forest, as

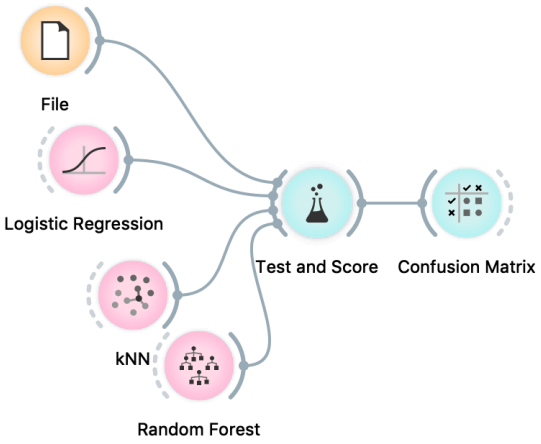
demonstrated in the now-familiar workflow with *Test & Score* widget.





There are, of course, cases where logistic regression fails miserably, and it is not hard to, for instance, paint a data set where the performance of this model would score much lower than random forests. Because of the simplicity and robustness that comes with it, logistic regression is an often-used technique. Logistic regression is also a core building block in neural networks, and we will address this briefly towards the end of this course.

Lesson 20: *k*-Nearest Neighbors



Here is one more technique before we finish with methods for classification. It is perhaps the most intuitive technique of them all, but surprisingly imprecise. We mention it here solely because of the concept, and not as a method you should use when developing a predictive model.

The method is called *k*-nearest neighbors, and it does what the name says. It classifies the new data instance according to its k nearest neighbors from the training set. For example, in the space with two continuous features, the data instance marked with A will be classified as blue, and instance B as red. We have used 3 for the value of *k*, that is, for every new instance, we found five closest data instances in the training set.

The parameter for k-nearest neighbors method is *k*, which is by default set to 10, and the distance function, where the default is Euclidean distance. In general, *k*-NN could use any other approach to distance scoring.

The performance of the nearest neighbor classification on heart disease data set is not stellar. Nearest neighbors perform substantially worse than logistic regression. In cases, where data would reside in two-dimensions, *k*-NN would do something similar to what we would be doing manually. Thus, this approach is at least conceptually interesting.