EMORY UNIVERSITY

Machine Learning II

Homework #3 **Due: turned in by Monday 03/23/2020 before class**

Carl Xi

(put your name above)

Total grade: _____ out of ___100___ points

Please answer the following questions and submit your assignment as a single PDF file by uploading it to the HW3 drop-box on the course website.

Hands-on predictive modeling (100 points)

Download the dataset on spam vs. non-spam emails from http://archive.ics.uci.edu/ml/datasets/Spambase. Specifically, (i) file "*spambase.data*" contains the actual data, and (ii) files "*spambase.names*" and "*spambase.DOCUMENTATION*" contain the description of the data. This dataset has 4601 records, each record representing a different email message. Each record is described with 58 attributes (indicated in the aforementioned *.names* file): attributes 1-57 represent various content-based characteristics extracted from each email message (related to the frequency of certain words or certain punctuation symbols in a message as well as to the usage of capital letters in a message), and the last attribute represents the class label for each message (spam or non-spam).

Task: The general task for this assignment is to build *two separate models* for detecting spam messages (based on the email characteristics that are given) using RapidMiner or any other tool you prefer (e.g., Python, Spark, etc.):

1. [Start with this task] The best possible model that you can build in terms of the *overall predictive accuracy*;
2. The best cost-sensitive classification model that you can build in terms of the *average misclassification cost*.

Some specific instructions for your assignment/write-up:

- Reading the data into RapidMiner (or your software of choice): Data is in a comma-separated-values (CSV) format. You may also want to add the attribute names (which are in *spambase.names* file) to the data file as the first line. It might be easiest to read data into Excel, save it as Excel file, and then import it to RapidMiner.
- Exploration: Make sure to explore multiple classification techniques. You should definitely consider decision trees, *k*-nearest-neighbors, and Naïve Bayes, but you are free to experiment with any other classification techniques you know (for example, you can try applying some meta-modeling techniques, etc.).
    - Also, explore different configurations of each technique (for example, you should try varying some key parameters, such as values of *k* for *k*-NN, etc.) to find which configurations work best for this application. You can use parameter optimization approaches to help you with that.
- Make sure to explore the impact of various data pre-processing techniques, especially normalization and attribute selection.
- When building cost-sensitive prediction models, use 10:1 cost ratio for different misclassification errors. (I think it should be pretty clear which of the two errors –

classifying a non-spam message as spam vs. classifying a spam message as non-spam – is the costlier one in this scenario.)

- In general, use best practices when evaluating the models: evaluate on validation/test data, discuss the confusion matrix and some relevant performance metrics (not just the required accuracy and average misclassification cost, but also precision, recall, f-measure – especially for your best/final models…), show some visual indications of model performance (such as ROC curves).
- Finally, as a deliverable, produce a write-up (i.e., a single PDF file) describing your aforementioned explorations. Report the performances of different models that you tried (i.e., using different data mining techniques, different attribute selection techniques, etc.) What was the performance of the best model in the cost-unaware task (i.e., in terms of accuracy)? What was the performance of the best model in the cost-aware task (i.e., in terms of expected cost)? Discuss the best models in two different tasks (as well as their performance) in detail, provide some comparisons. Draw some conclusions from the assignment.

Evaluation: 100 points total.

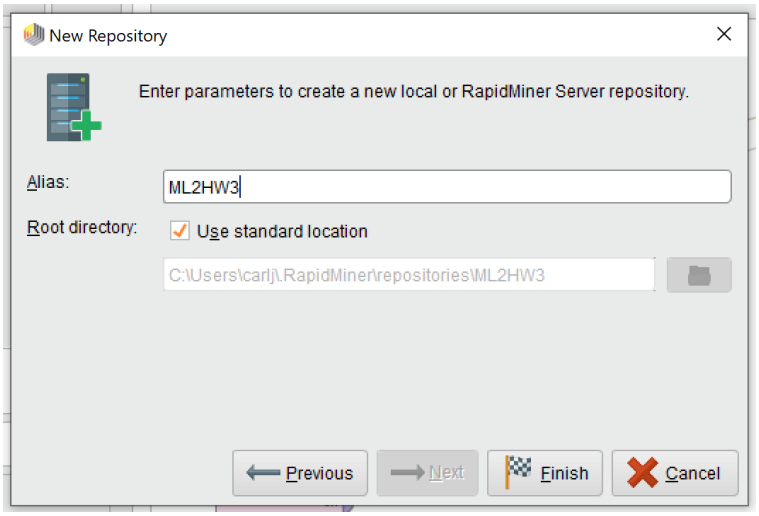
- Performance: 35 points (based on the performance achieved by your best reported models).
- Exploration/write-up: 65 points (based on the comprehensiveness of your exploration, i.e., when searching for the best performing model, did you evaluate and report just one or two techniques, or did you try a number of different variations, based on what you know from the class?).

Writeup:

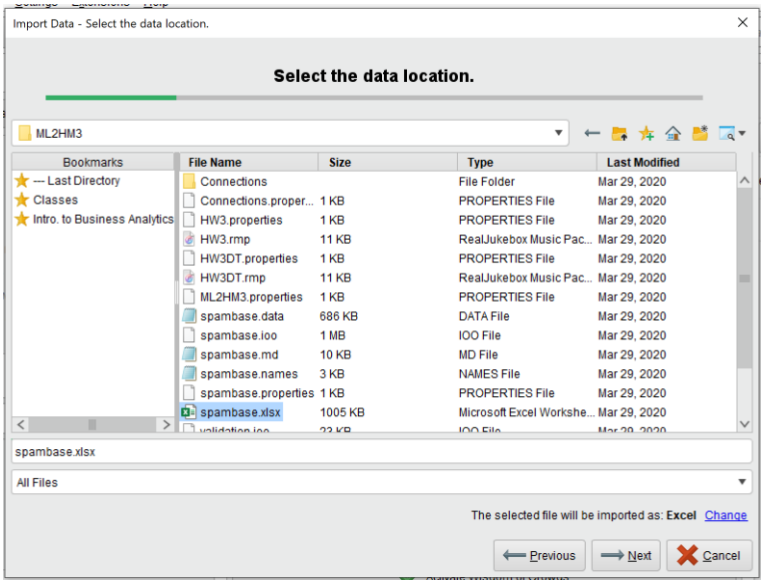
To keep things simple, I have decided to conduct all my work in RapidMiner. This includes data exploration, cleaning, exploratory data analysis, modeling, cross-model comparison, and final model selection. This entire process is repeated twice for the best model with the best overall predictive accuracy, as well as the best cost-sensitive classification model with the best average misclassification cost.

Prior to our RapidMiner work, we download and import both the spambase.data and spambase.names files into excel, append the headers into the top row of the dataset, and export the merged file as an excel file.

We start by creating a new repository in RapidMiner. The screenshots below show this process:



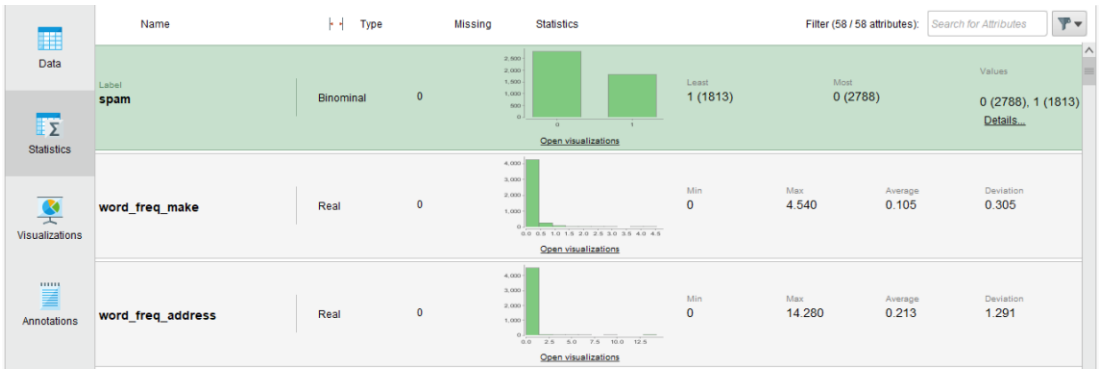
With our repository set up, we then move the excel file from above into our repository folder. We then import our data using the gigantic ‘import data’ button:



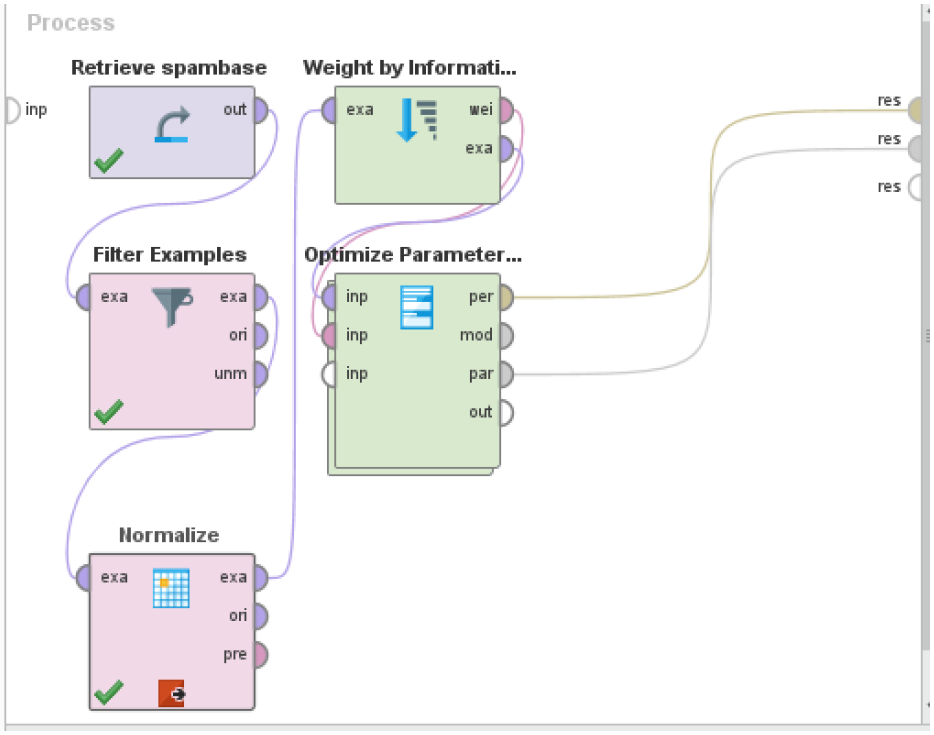
The process is pretty intuitive, we choose our excel file, but make sure to set the spam column as binomial and the label column (target variable). We lastly make sure to save our database in our repository.

	char_freq_... *real*	char_freq_... *real*	capital_run... *real*	capital_run... *integer*	capital_run... *integer*	spam *binominal label*
1	0.000	0.000	3.756	61	278	1
2	0.180	0.048	5.114	101	1028	1
3	0.184	0.010	9.821	485	2259	1
4	0.000	0.000	3.537	40	191	1

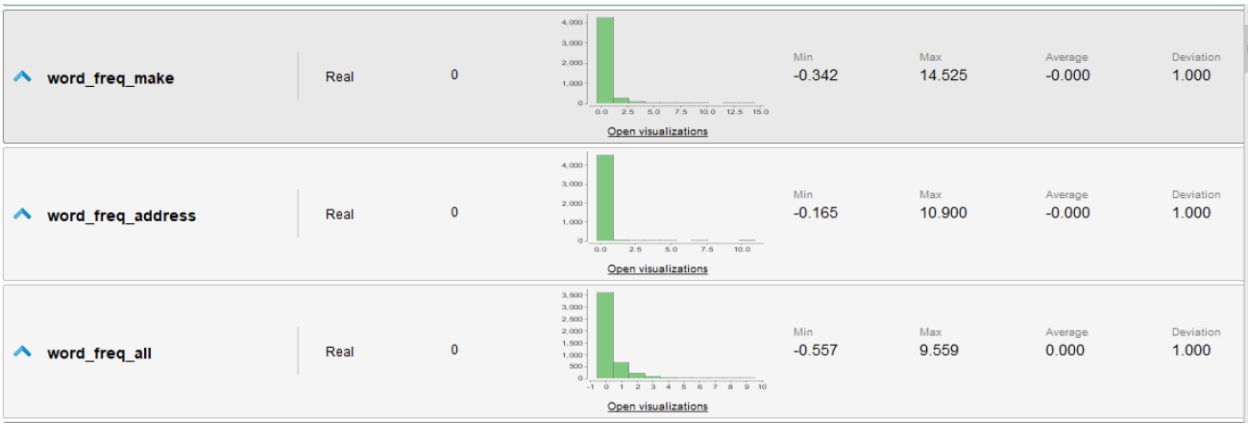
We can now look at our raw data. We see that our binary target variable is reasonably balanced, so there isn't as much of a need to rebalance. However, we see two problems with our variables. First, most of them are heavily skewed, with extreme outliers. Second, they are on very different scales, with variables like capital_run_length_total ranging from 1 to 15841 and variables like char_freq_brackets ranging from 0 to 4.081. The former can be solved with transformations like log or exponents, while the latter can be solved with normalization. Unfortunately, we are unfamiliar with transformation in RapidMiner, so we will only conduct normalization.



With the data explored and problems identified, we then build the mockup of a fundamental k-NN model for testing. While not necessary, we included a filter that filters out all NAs from our target variable just for good practice. The normalize module will automatically normalize all our variables to reduce data bias. We add a breakpoint after the 'normalize' module and run once to review the cleaned final data that we will be using for modeling.



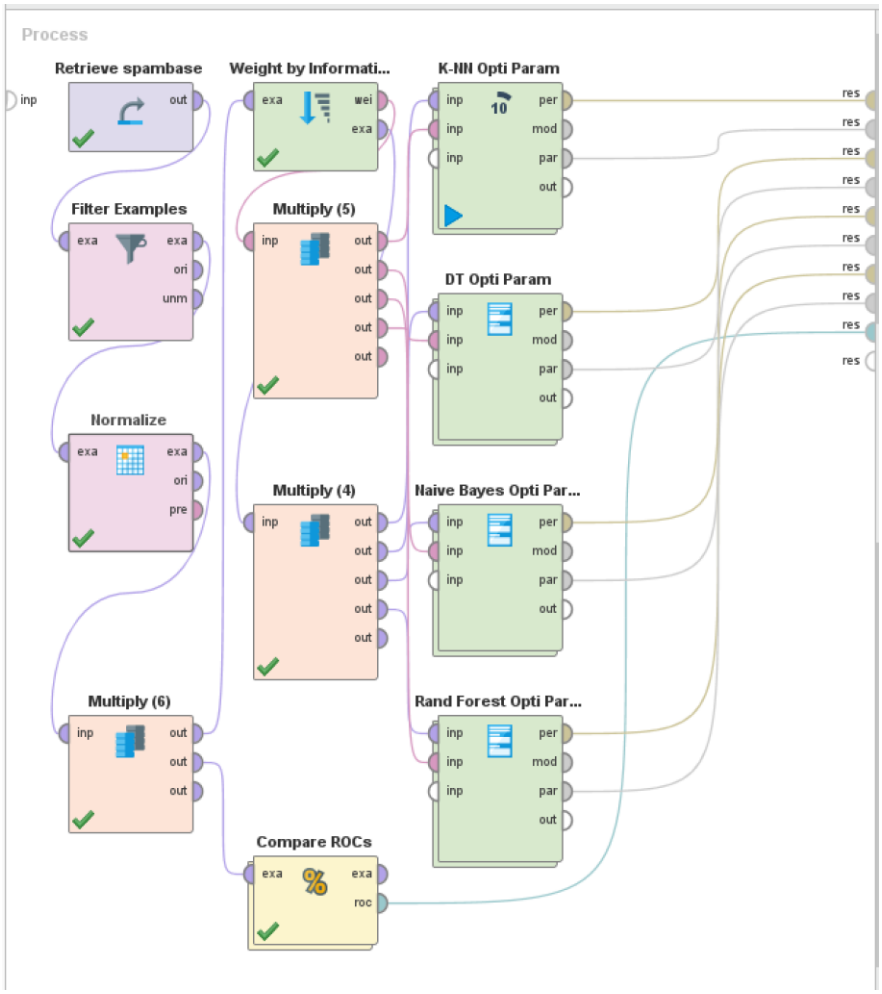
While still skewed, our normalized data is looking much better across the variables. This data is acceptable given our proficiency in RapidMiner. We are ready to proceed.



To keep things simple, we will be comparing only the results of k-NN, decision trees, Naïve Bayes and random forest (representing meta-modeling) for this assignment. Note that our process can be repeated for any number and type of models, and the swap process is very simple (simply edit the compare ROC module).

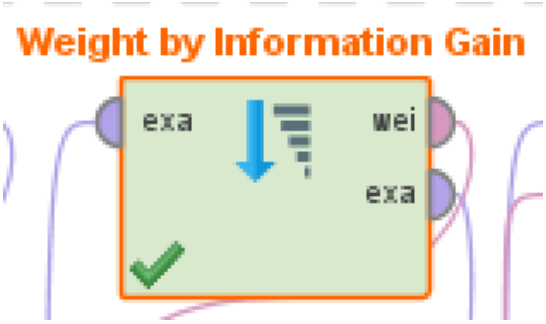
To confirm our model selection, we conferred with the https://mod.rapidminer.com/#app website with the features of our dataset. It seems like our decision is correct

To save your time and mine, I have compressed all 8 models into 1.



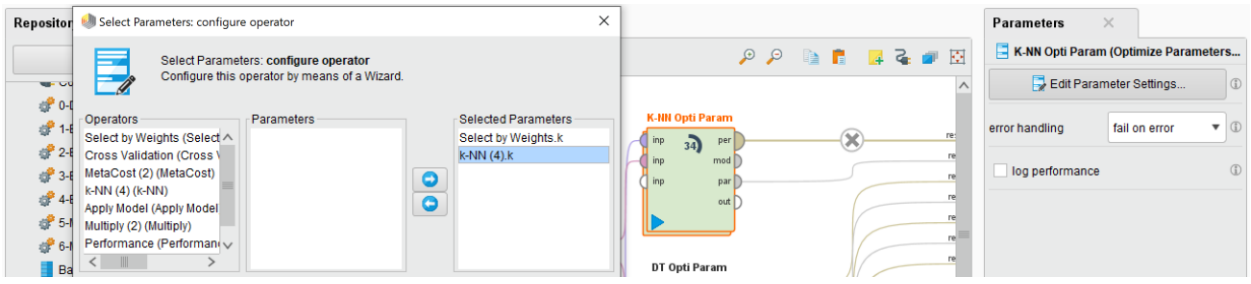
The screenshot above calculates both performance indicators (accuracy, precision. Recall, f-measure), and cost-sensitivity using the cost matrix. As well, it generates ROC curves of each model after parameter tuning, as well as a cross-model ROC curve comparison chart without parameter tuning as a baseline.

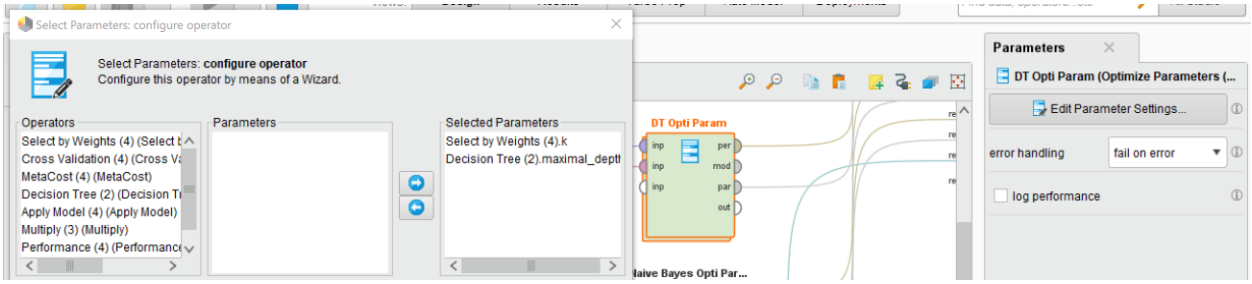
Parameter tuning is done by the following. We first weigh all 57 variables by information gain:

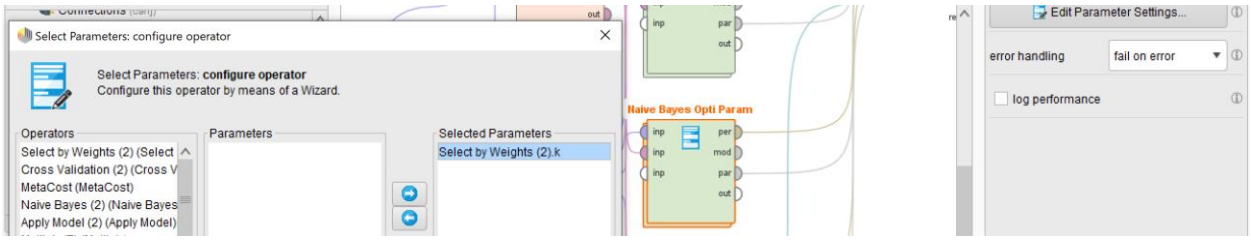


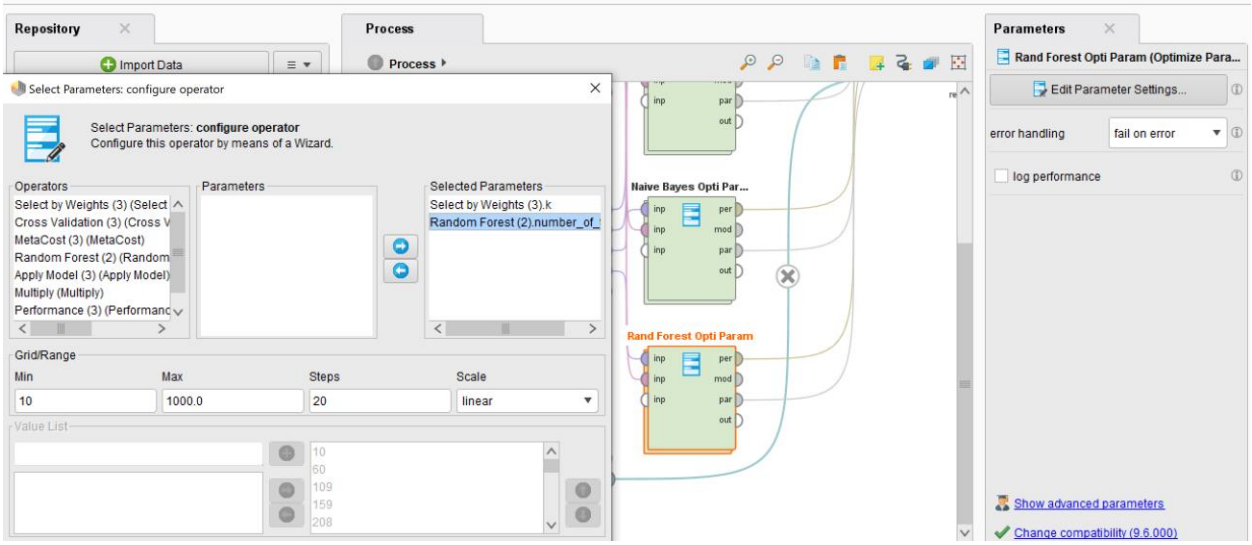
The data is then passed onto the four optimization parameter modules for our four models. Each Optimization parameter module optimizes two aspects. First, each module will optimize the number of variables and which variables to chose (feature selection) using the weights generated by the weight module above. Each module also will tune one parameter of the respective models. We tuned number of neighbors for k-NN, number of trees for random forest, and max depth for decision tree. Naïve Bayes do not have any parameters to tune. This alone took forever, so we only tune one variable to demonstrate our capability. More variables can be added to be tuned with more computational power and time.

The following 4 screenshots showcase the parameter tuning of each of the 4 models.

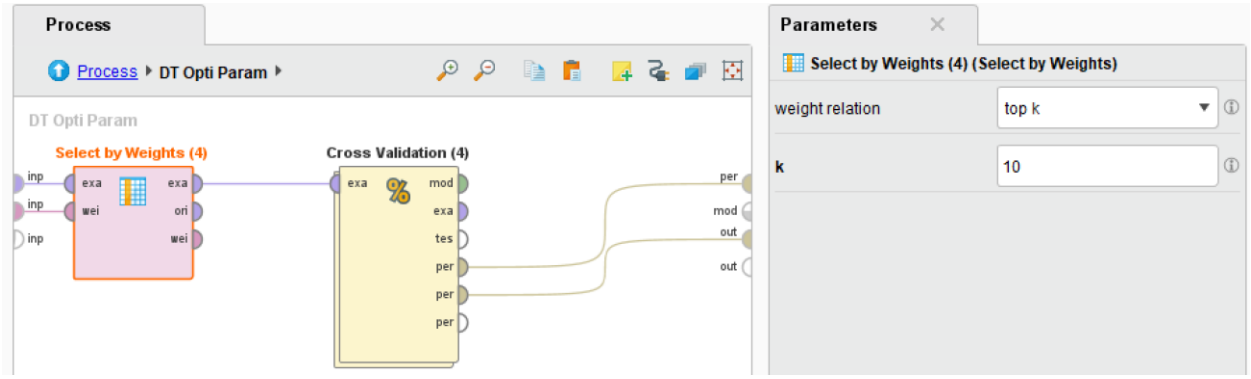




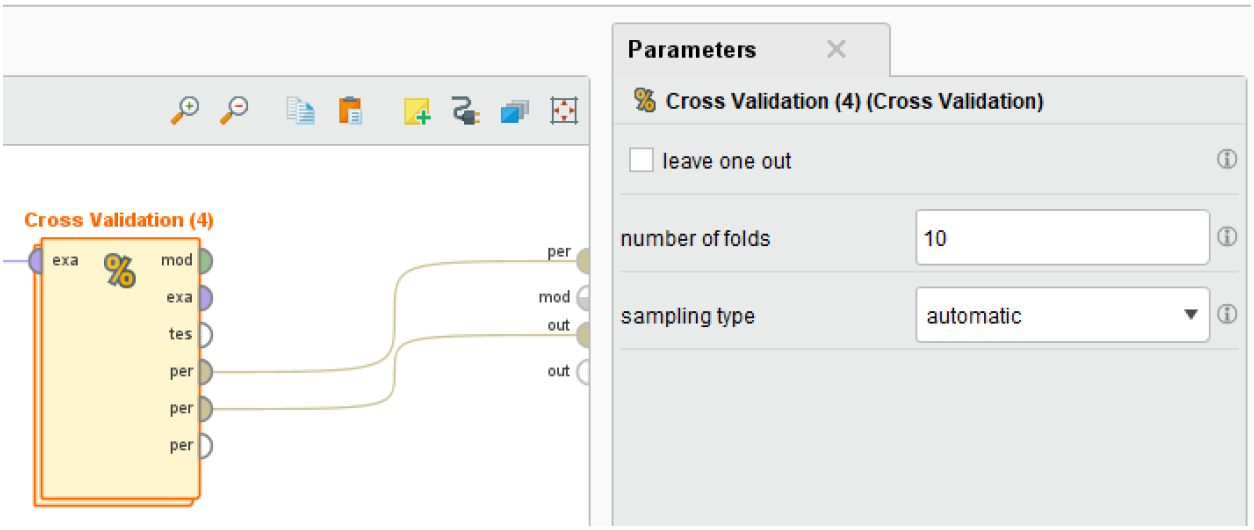




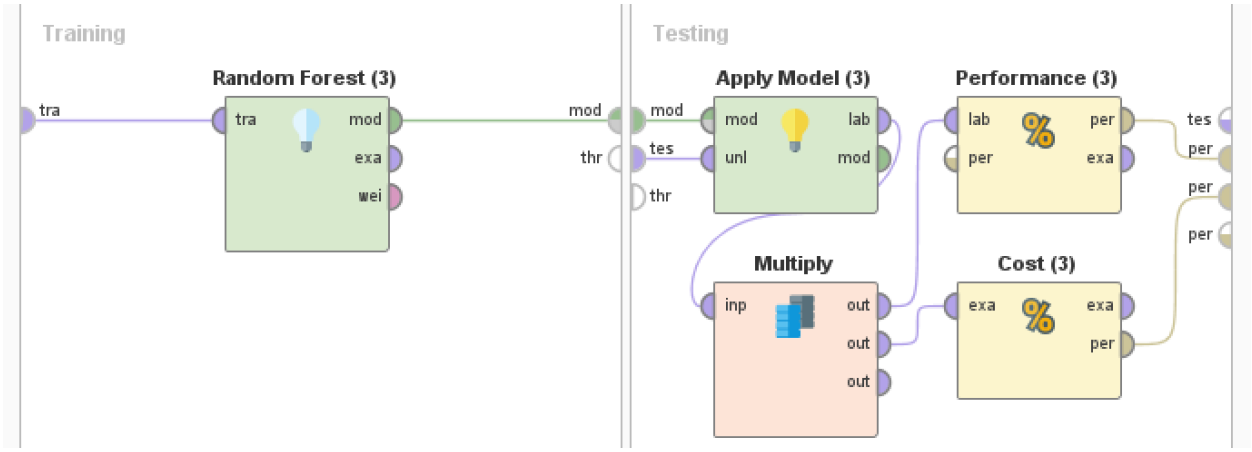
The insides of each optimization parameter module is the same with exception to the model being used. The first layer we have the ‘select by weights’, which chooses the top ‘k’ variables to use for each model. The variables are then passed onto the cross-validation module.



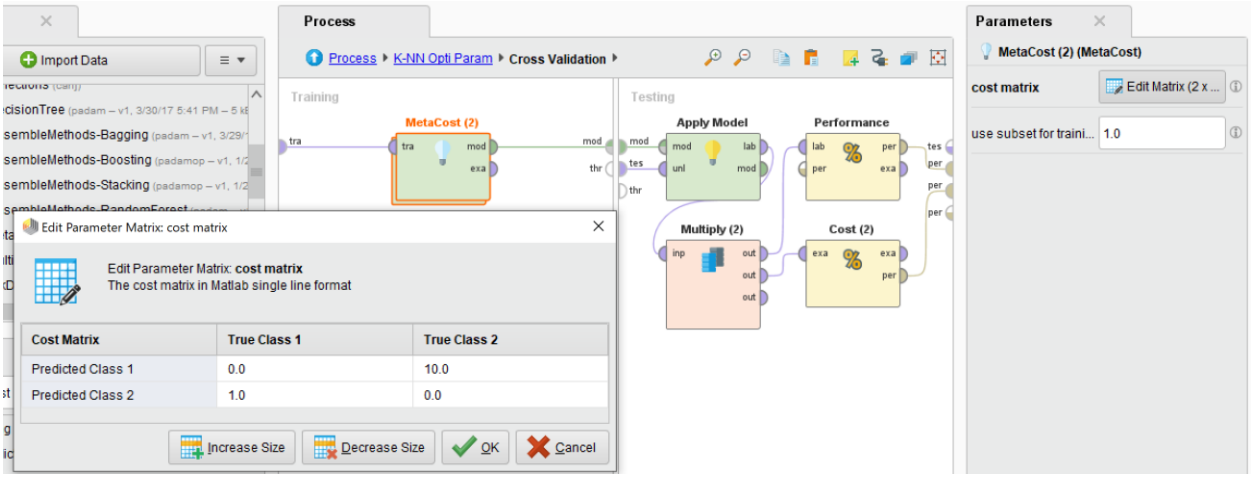
The cross-validation module is 10-fold with automatic shuffling. There are arguments for both stratified and random shuffling, so I decided to let the system choose the most appropriate one, especially since our dataset is thankfully relatively simple and small.



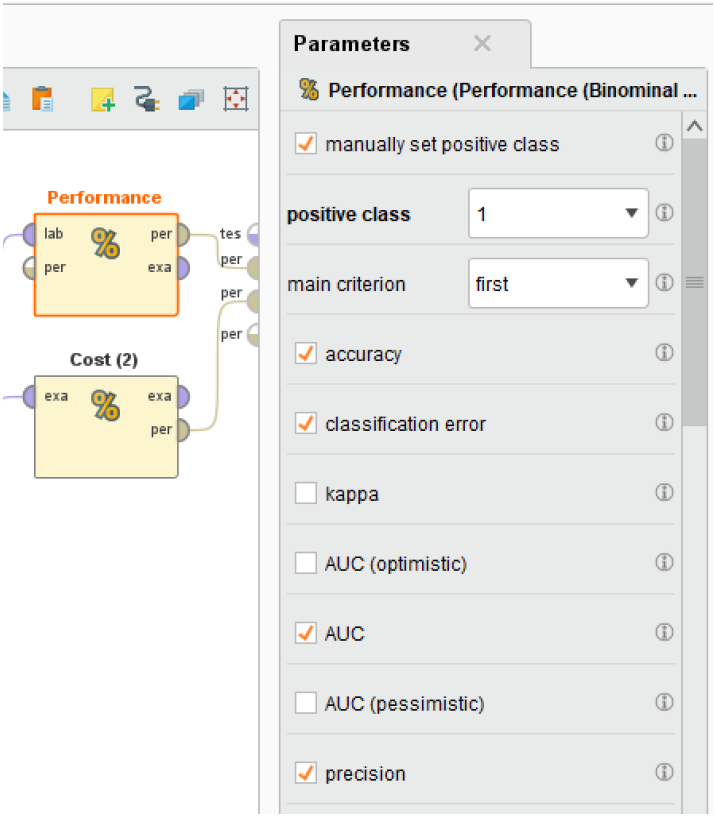
Opening our Cross-Validation module, we see that the data is passed onto the model. The first 4 models will be run with just the model itself.



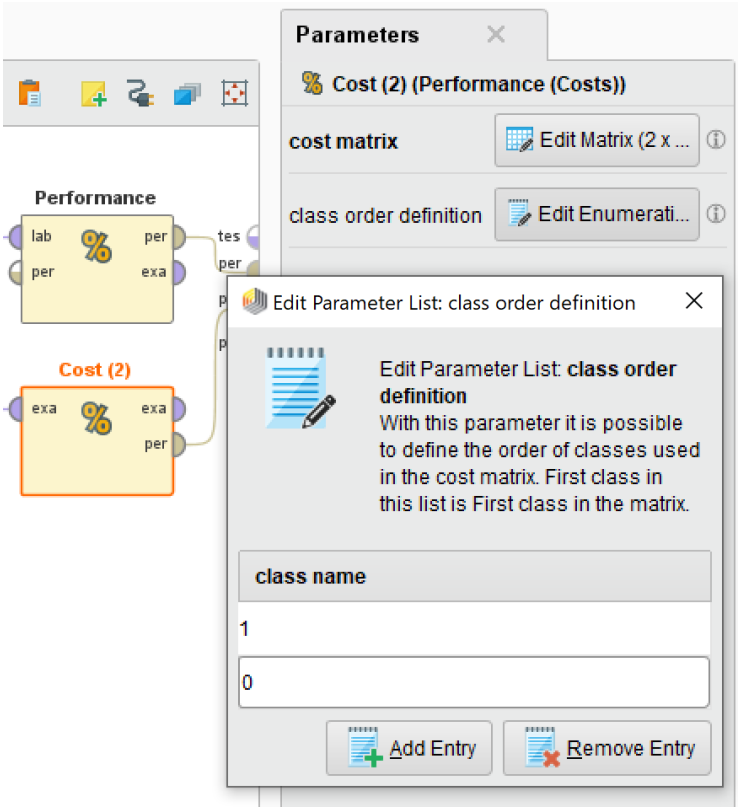
The second 4 will be run with the model encased within a metacost module. Within it, one of the four models is present. The data passes through the model and is applied to the performance and cost calculations. The performance module calculates all the performance indicators mentioned above, while the cost module calculates the cost-sensitivity of our prediction model, with False Positive (not-spam marked as spam) having a cost of 10, while False Negative (spam marked as not spam) having a cost of 1. We can see why this is the case (important emails being flagged vs a spam email showing up in your inbox).



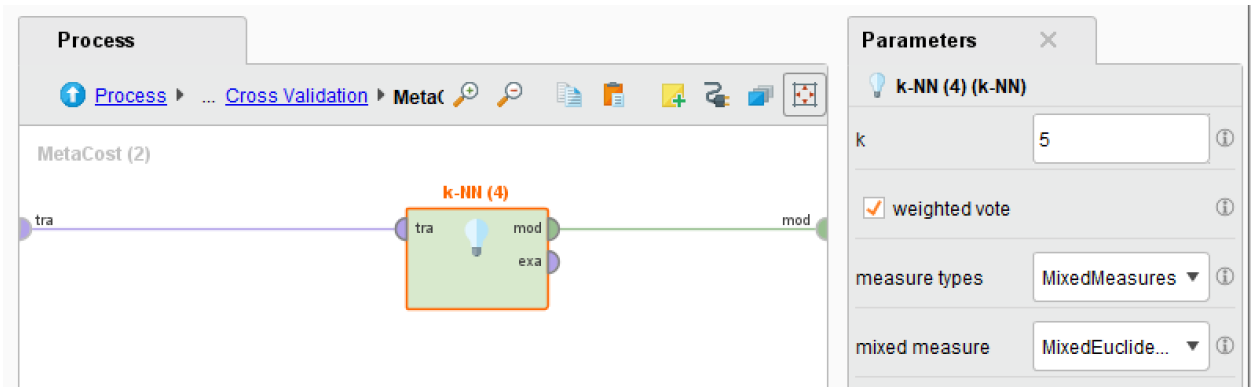
The variables that performance is putting out (as well as recall and f-measure not shown). Note that we manually set positive class as 1:



The cost matrix for our cost calculations is the same as the one for the metacost module. We make sure to set the class order as 1 (spam) first and 0 (not spam) second, to be inline with all our other modules (e.g. the performance calculations).



Finally, opening up the MetaCost module we see the actual model. In this case, it's k-NN. This entire Matryoshka doll-esque nested modules is repeated four times for the four models, and it's only here on the model level that the difference occurs.



I won't post screenshots of this module for all four models, but you get the idea. The k shown above is arbitrary as our parameter tuning optimizer above will automatically select the best one. THIS IS FOR FINDING THE BEST COST EFFECTIVE MODEL. We will repeat the entire process 4 more times, but with metacost replacing the 4 models in each instance.

With everything setup, the rest was to hit run and wait for the process to run. It takes a few hours for everything to complete, as there is many layers of parameter tuning as well as feature selection for multiple models. The performance results are below:

	k-NN	Decision Tree	Naïve Bayes	Random Forest
Accuracy	91.72% +/- 1.61%	91.15% +/- 1.18%	89.11% +/- 1.84%	92.24% +/- 1.16%
Classification Error	8.28% +/- 1.61%	8.85% +/- 1.18%	10.89% +/- 1.84%	7.76% +/- 1.16%
AUC	0.962 +/- 0.016	0.920 +/- 0.017	0.937 +/- 0.011	0.976 +/- 0.006
Precision	91.53% +/- 2.02%	92.07% +/- 1.60%	86.92% +/- 2.43%	95.81% +/- 1.59%
Recall	87.04% +/- 2.52%	84.89% +/- 2.71%	85.22% +/- 3.07%	84.01% +/- 2.58%
F-Measure	89.22% +/- 2.14%	88.31% +/- 1.64%	86.04% +/- 2.37%	89.50% +/- 1.67%
Misclassification Cost	0.368 +/- 0.083	0.349 +/- 0.064	0.565 +/- 0.105	0.209 +/- 0.059
# Parameters	13	9	29	47
Tuned param	11	Max Depth: 18	N/A	200 Trees

The Confusion matrix of the four models are below:

k-NN:

	true 1	true 0	class precision
pred. 1	1578	146	91.53%
pred. 0	235	2642	91.83%
class recall	87.04%	94.76%	

Decision Tree:

	true 1	true 0	class precision
pred. 1	1539	133	92.05%
pred. 0	274	2655	90.65%
class recall	84.89%	95.23%	

Naïve Bayes:

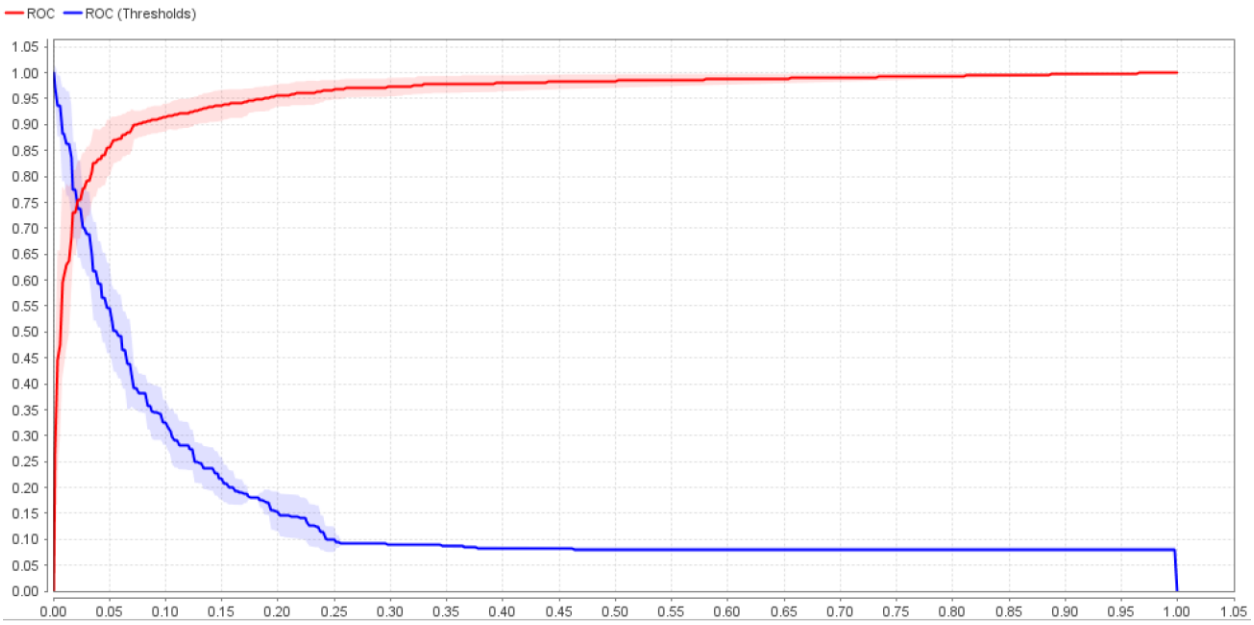
	true 1	true 0	class precision
pred. 1	1545	233	86.90%
pred. 0	268	2555	90.51%
class recall	85.22%	91.64%	

Random Forest:

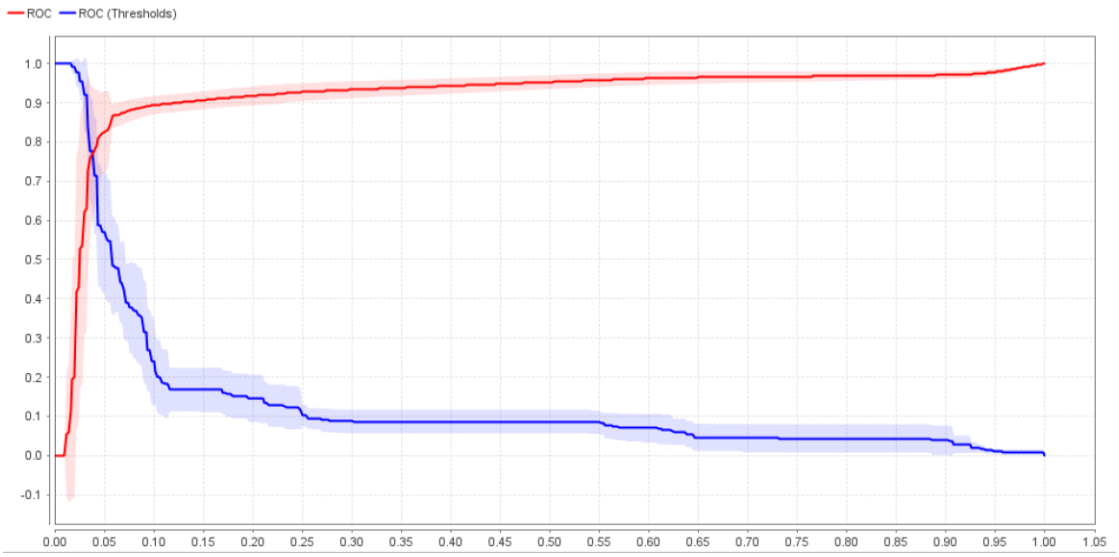
	true 1	true 0	class precision
pred. 1	1523	67	95.79%
pred. 0	290	2721	90.37%
class recall	84.00%	97.60%	

The ROC curves of the four models are below:

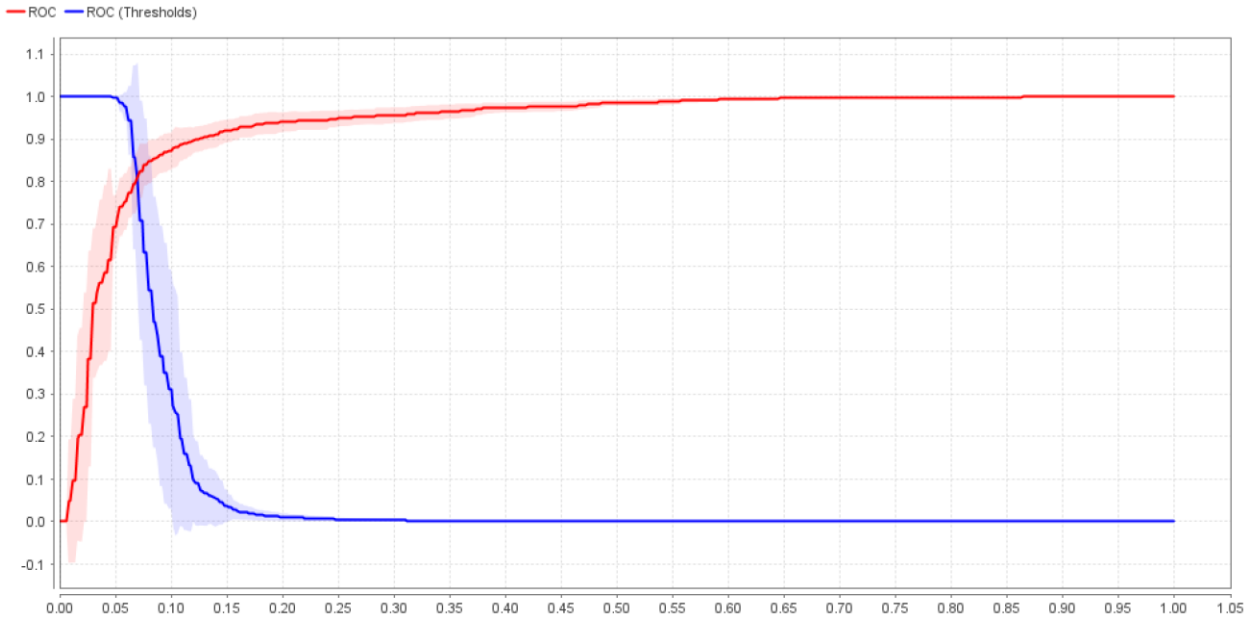
k-NN:



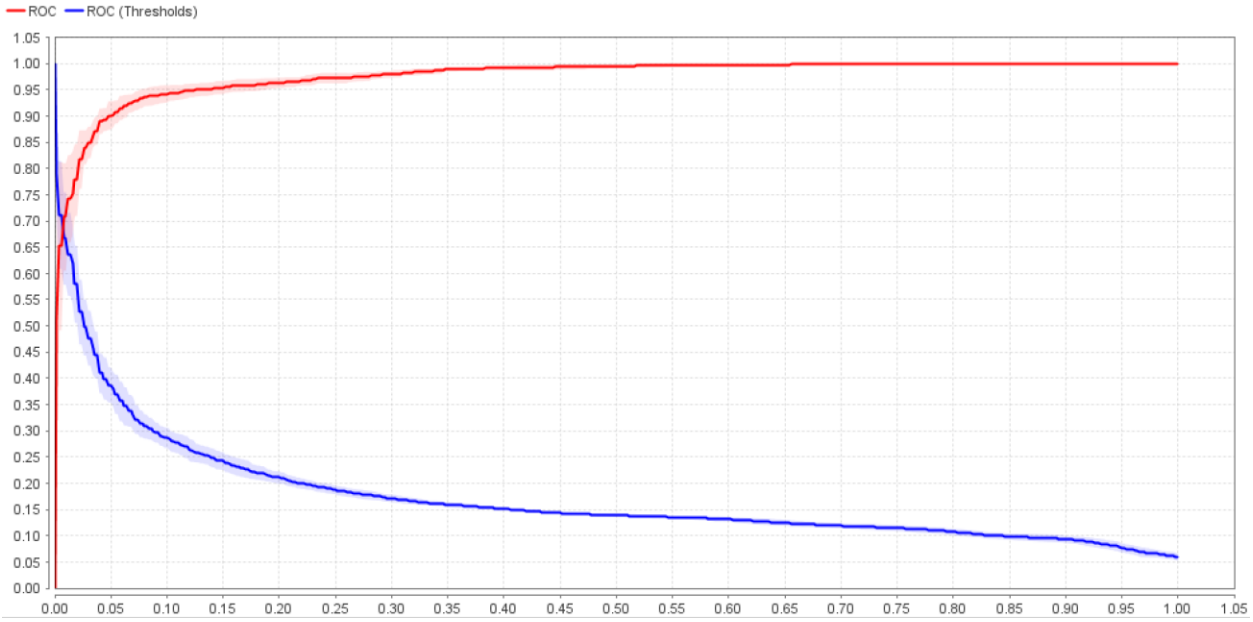
Decision Tree:



Naïve Bayes:



Random Forest:



This entire process is repeated again with metacost replacing the 4 models in the formulas above:

	k-NN	Decision Tree	Naïve Bayes	Random Forest
Accuracy	88.85% +/- 1.47%	85.11% +/- 0.92%	90.46% +/- 1.09%	73.29% +/- 0.69%
Classification Error	11.15% +/- 1.47%	14.89% +/- 0.92%	9.54% +/- 1.09%	26.71% +/- 0.69%
AUC	0.936 +/- 0.013	0.951 +/- 0.010	0.949 +/- 0.009	0.972 +/- 0.008
Precision	94.96% +/- 1.08%	97.97% +/- 1.21%	88.25% +/- 2.35%	99.32% +/- 1.18%
Recall	75.73% +/- 3.78%	63.54% +/- 2.20%	87.54% +/-3.07%	32.43% +/- 1.67%
F-Measure	84.21% +/- 2.39%	77.06% +/- 1.70%	87.84% +/-1.46%	48.88% +/- 1.96%
Misclassification Cost	0.254 +/- 0.038	0.196 +/- 0.032	0.512 +/- 0.101	0.275 +/- 0.018
# Parameters	14	32	31	30
Tuned param	2- NN	Max Depth: 19	N/A	100 Trees

The Confusion matrix of the four models are below:

k-NN:

	true 1	true 0	class precision
pred. 1	1373	73	94.95%
pred. 0	440	2715	86.05%
class recall	75.73%	97.38%	

Decision Tree:

	true 1	true 0	class precision
pred. 1	1152	24	97.96%
pred. 0	661	2764	80.70%
class recall	63.54%	99.14%	

Naïve Bayes:

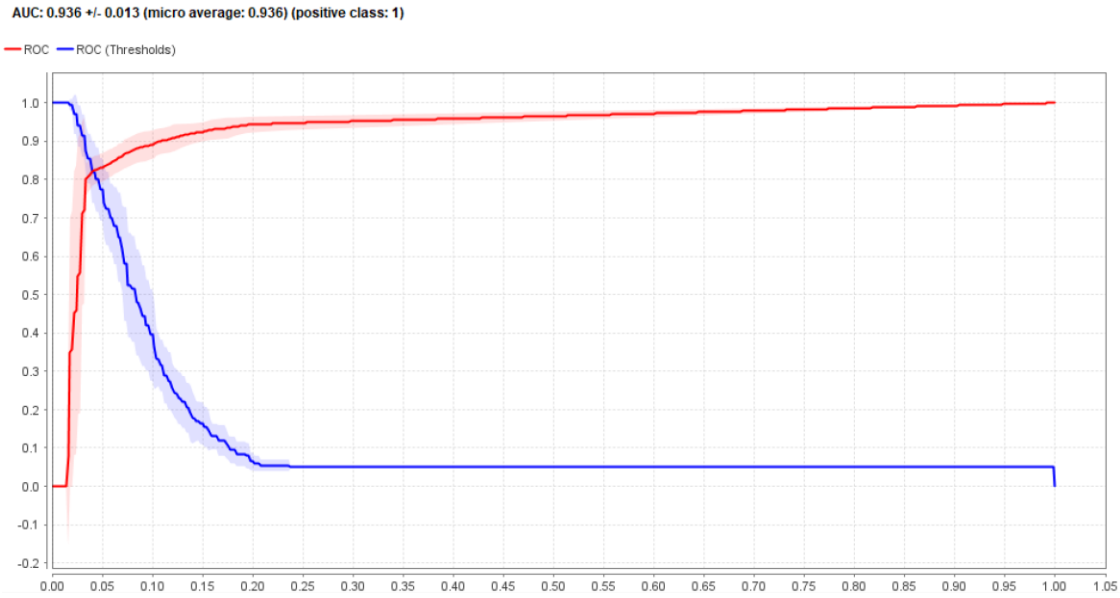
	true 1	true 0	class precision
pred. 1	1587	213	88.17%
pred. 0	226	2575	91.93%
class recall	87.53%	92.36%	

Random Forest:

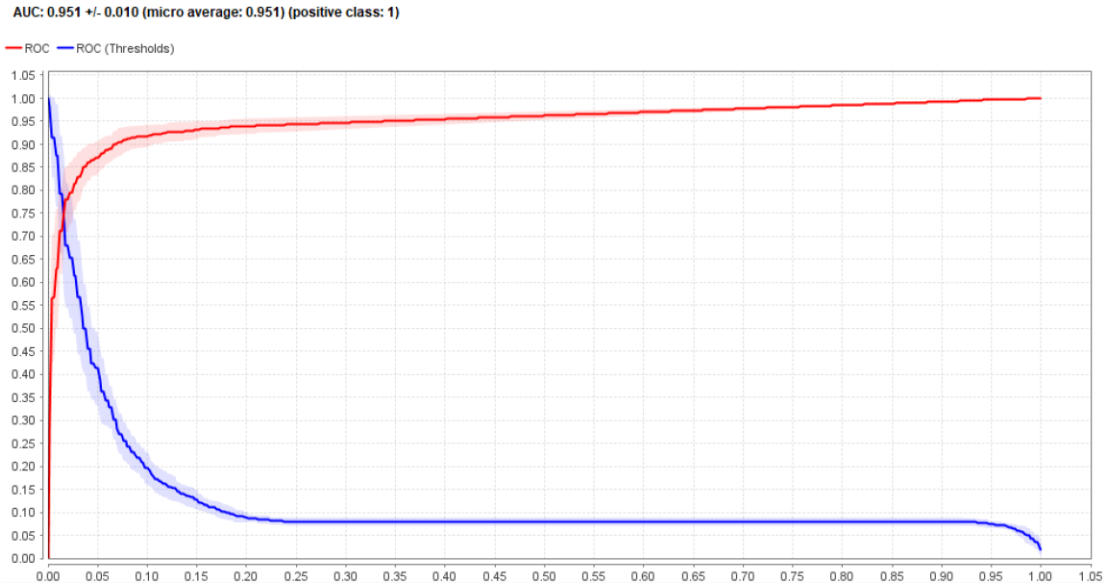
	true 1	true 0	class precision
pred. 1	588	4	99.32%
pred. 0	1225	2784	69.44%
class recall	32.43%	99.86%	

The ROC curves of the four models are below:

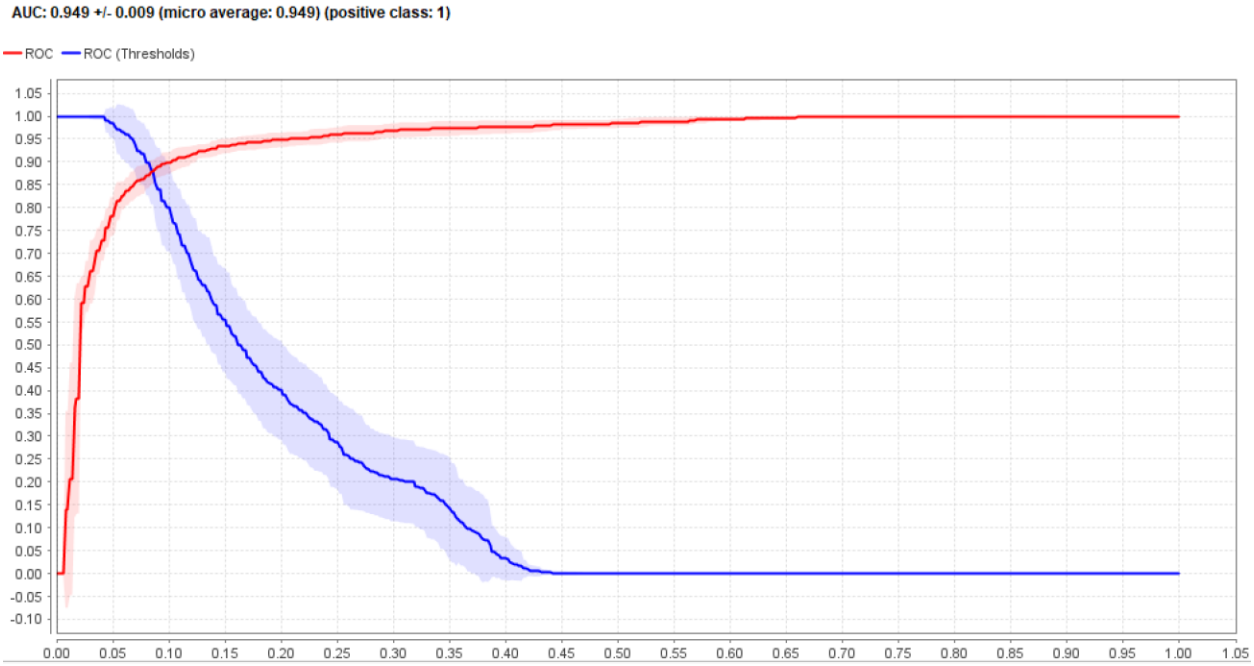
k-NN:



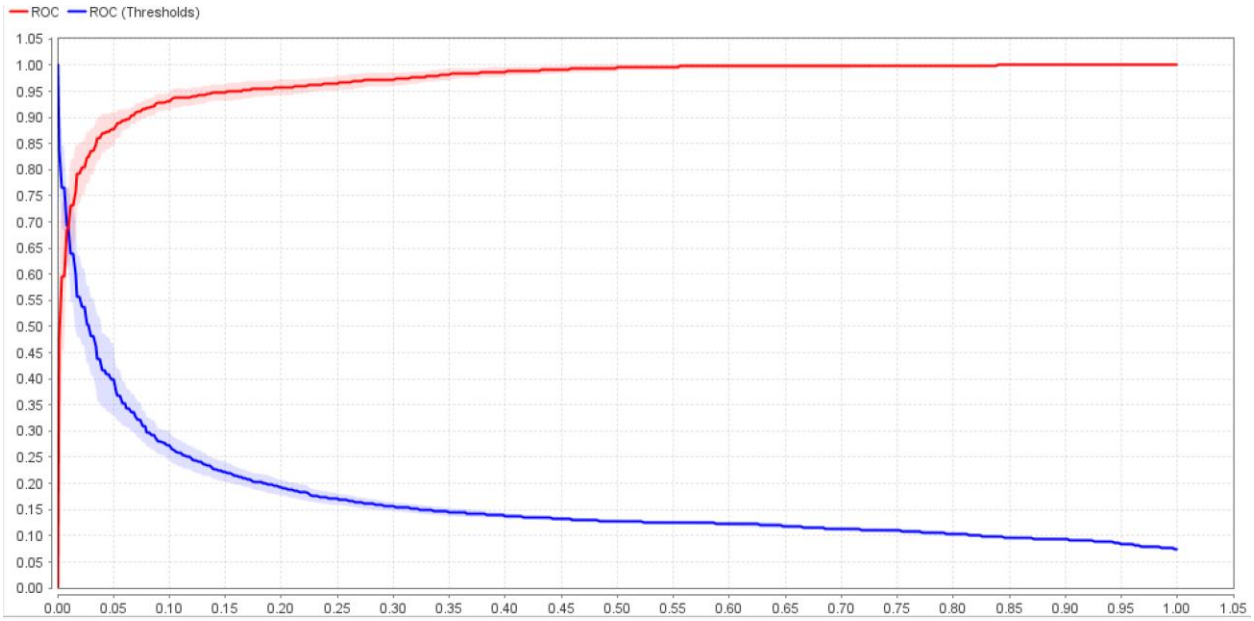
Decision Tree:



Naïve Bayes:



Random Forest:



Overall: we can see that Random Forest has the best accuracy by far among the first set of 4 models at 92.24% +/- 1.16%, while the Decision Tree has the best misclassification performance among the second set of 4 models at 0.196 +/- 0.032.

- Key takeaways: It seems like normalization affected the kNN model the most, which is inline with what we have learned in class. kNN is heavily influenced by distance, which is why this normalization is especially helpful here. With that said, it is not as helpful for decision tree and naïve bayes, but do not really hurt them, which is why I decided to keep them in all four models.
- While decision tree has automatic feature selection, the feature selection step of our process was valuable for Naïve Bayes and kNN. Again kNN considers the distance of all variables, so having the most valuable variables definitely help.
- As our second task is to identify the most cost-effective model, having a less True Positives becomes more important than having a better accuracy. Indeed, we see that the best cost-effective model trades some accuracy in place of better positive recall when compared to the most accurate model. This is sure to change as we manipulate the cost matrix (currently 1:10 for FN to FP), but generally for binary classification we should expect to see a similar pattern for similar classification tasks.