EMORY UNIVERSITY

Machine Learning II

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

Qiyu Wang

(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.

  The impact of normalization is critical for some techniques such as KNN.

- 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). It is relatively easy to achieve an accuracy greater than 90% and/or an average cost of less than 0.2.
- 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? You should explore at least one basic classification technique and one meta-modeling technique as well as the impact of normalization and feature selection, and report the predictive accuracy and average misclassification cost).

1. Importing the Data

```
df = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data",header=None)
```

```
from urllib.request import urlopen

label = urlopen("http://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names").read().decode('utf-8')
```

The data and attribute names are in .data format independently, so I used Python to read in them and then I transformed the spambase.names into a usable format.

```
# formatting the label

label = label.split("non-spam classes\r\n\r\n")[1]
label = label.split("continuous.\r\n")
for i in range(len(label)):
    label[i] = label[i].split(":")[0]
label[-1] = "spam"
```

```
label
```

```
['word_freq_make',
 'word_freq_address',
 'word_freq_all',
 'word_freq_3d',
```

```
df.columns = label
```

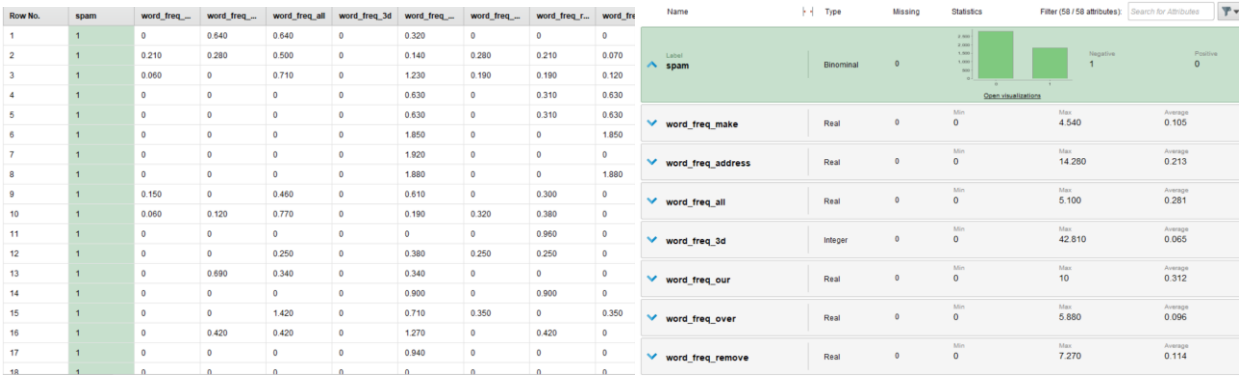
```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
word_freq_make	4601.0	0.104553	0.305358	0.0	0.000	0.000	0.000	4.540
word_freq_address	4601.0	0.213015	1.290575	0.0	0.000	0.000	0.000	14.280
word_freq_all	4601.0	0.280656	0.504143	0.0	0.000	0.000	0.420	5.100
word_freq_3d	4601.0	0.065425	1.395151	0.0	0.000	0.000	0.000	42.810
word_freq_our	4601.0	0.312223	0.672513	0.0	0.000	0.000	0.380	10.000
word_freq_over	4601.0	0.095901	0.273824	0.0	0.000	0.000	0.000	5.880
spam	4601.0	0.394045	0.488698	0.0	0.000	0.000	1.000	1.000

Then I assigned the names to the column names, and output a summary stats to check the distributions of attributes. The data have 39.4% of spam which does not contribute to any imbalance issue. So, I output the dataframe as a csv to be read into RapidMiner.

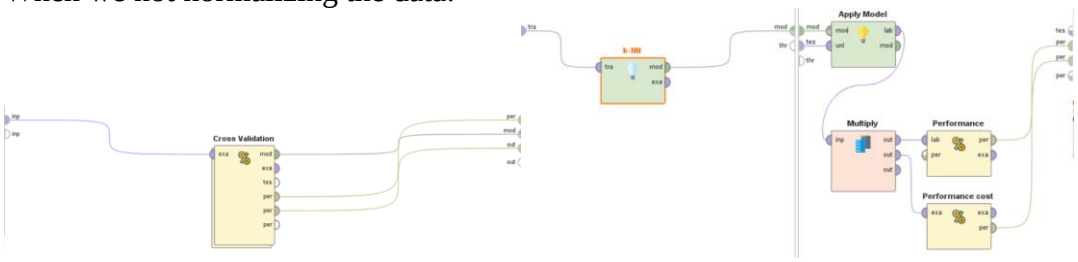
```
df.to_csv("C:/Users/10331/OneDrive/Desktop/spambase.csv", index = False)
```

Here I loaded it into the RapidMiner and checked the summary statistics to make sure everything is correct.



Row No.	spam	word_freq_...	word_freq_...	word_freq_all	word_freq_3d	word_freq_...	word_freq_...	word_freq_r...	word_fre
1	1	0	0.640	0.640	0	0.320	0	0	0
2	1	0.210	0.280	0.500	0	0.140	0.280	0.210	0.070
3	1	0.060	0	0.710	0	1.230	0.190	0.190	0.120
4	1	0	0	0	0	0.630	0	0.310	0.630
5	1	0	0	0	0	0.630	0	0.310	0.630
6	1	0	0	0	0	1.850	0	0	1.850
7	1	0	0	0	0	1.920	0	0	0
8	1	0	0	0	0	1.880	0	0	1.880
9	1	0.150	0	0.460	0	0.610	0	0.300	0
10	1	0.060	0.120	0.770	0	0.190	0.320	0.380	0
11	1	0	0	0	0	0	0	0.960	0
12	1	0	0	0.250	0	0.380	0.250	0.250	0
13	1	0	0.690	0.340	0	0.340	0	0	0
14	1	0	0	0	0	0.900	0	0.900	0
15	1	0	0	1.420	0	0.710	0.350	0	0.350
16	1	0	0.420	0.420	0	1.270	0	0.420	0
17	1	0	0	0	0	0.940	0	0	0
18	1	0	0	0	0	0	0	0	0

2. Normalizing the data
   Normalizing the data presented as important step for some algorithms but not for others. Here we tried KNN, where k = 4 selected by grid search:
   a. When we not normalizing the data:



   (those two pictures shows the inside of the grid search)
   The KNN model returns an accuracy of 82.79% and misclassification cost of 0.951.

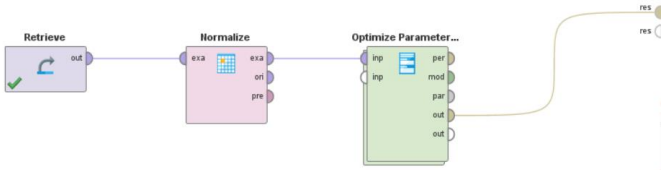
accuracy: 82.79% +/- 2.12% (micro average: 82.79%)

	true 1	true 0	class precision
pred. 1	1419	398	78.10%
pred. 0	394	2390	85.85%
class recall	78.27%	85.72%	

Misclassificationcosts

```
Misclassificationcosts: 0.951 +/- 0.147 (micro average: 0.951)
```

   b. After we normalized the data:



   The KNN model's accuracy increased to 91.61% and misclassification cost of 0.403.

accuracy: 91.61% +/- 1.00% (micro average: 91.61%)

	true 1	true 0	class precision
pred. 1	1590	163	90.70%
pred. 0	223	2625	92.17%
class recall	87.70%	94.15%	

Misclassificationcosts

```
Misclassificationcosts: 0.403 +/- 0.109 (micro average: 0.403)
```

   c. However, for other algorithms, such as decision trees and logistic regression, normalization does not impact the performance of models:

accuracy: 89.37% +/- 1.49% (micro average: 89.37%)

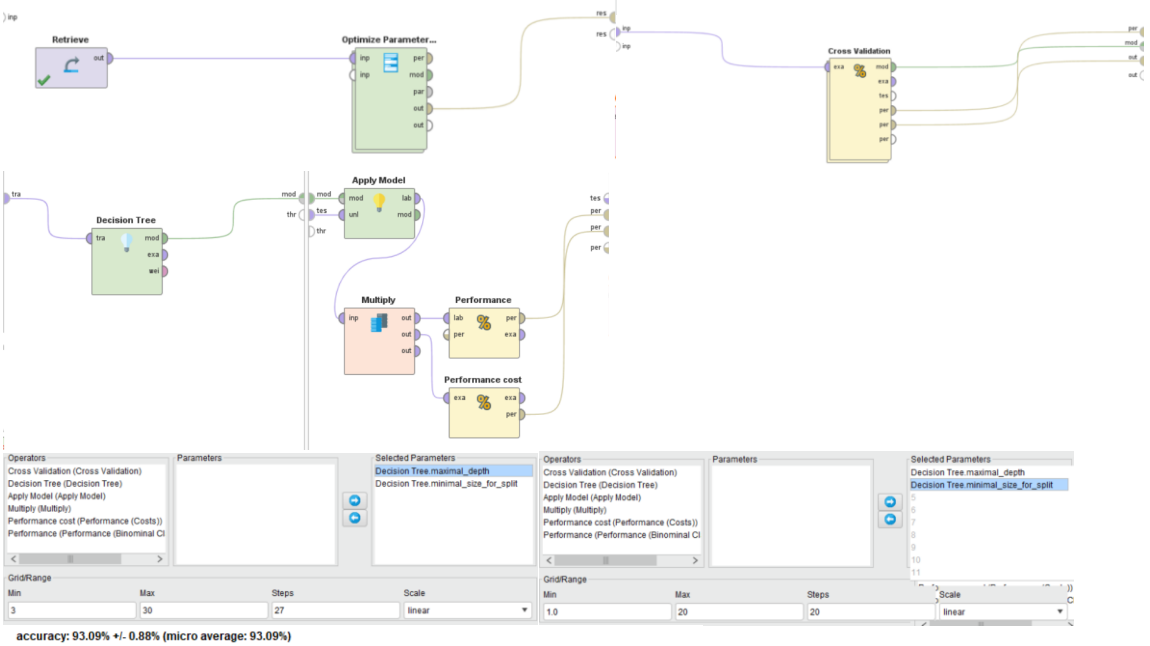
	true 1	true 0	class precision
pred. 1	1441	117	92.49%
pred. 0	372	2671	87.78%
class recall	79.48%	95.80%	

accuracy: 89.35% +/- 1.45% (micro average: 89.35%)

	true 1	true 0	class precision
pred. 1	1441	118	92.43%
pred. 0	372	2670	87.77%
class recall	79.48%	95.77%	

3. Grid Search

We used grid search for selecting best hyperparameter. Here we used decision tree as a example to show in internal structure. Best model is using information gain as criterion, a 23 depth and 16 min size for split decision tree, where the accuracy is at 93.26%.



accuracy: 93.09% +/- 0.88% (micro average: 93.09%)

	true 1	true 0	class precision
pred. 1	1642	147	91.78%
pred. 0	171	2641	93.92%
class recall	90.57%	94.73%	

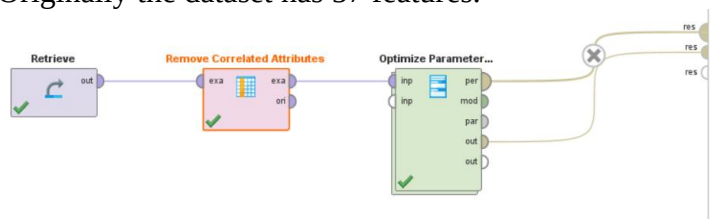
iteration	Decisio...	Decisio...	acc... ↓
170	19	11	0.931
105	18	7	0.930
184	17	12	0.929
296	17	19	0.929
153	18	10	0.929
245	14	16	0.929
241	10	16	0.929
182	15	12	0.929
253	22	16	0.929
197	14	13	0.929
94	23	6	0.928
187	20	12	0.928
233	18	15	0.928
134	15	9	0.928
99	12	7	0.928
271	24	17	0.928
160	25	10	0.928
185	18	12	0.928
201	18	13	0.928

4. Feature selections
   a. Remove correlated attributes

      Here we used the decision tree model from last section as a baseline, and we tried removing highly correlated terms.

ExampleSet (4,601 examples, 1 special attribute, 57 regular attributes)

Originally the dataset has 57 features.



By excluding features that have higher than 0.9 correlation, we exclude 1 feature. The performances not affected much.

ExampleSet (4,601 examples, 1 special attribute, 56 regular attributes)

accuracy: 93.18% +/- 0.92% (micro average: 93.18%)

	true 1	true 0	class precision
pred. 1	1668	169	90.80%
pred. 0	145	2619	94.75%
class recall	92.00%	93.94%	

By excluding features that have higher than 0.7 correlation, we exclude 2 features.

ExampleSet (4,601 examples, 1 special attribute, 55 regular attributes)

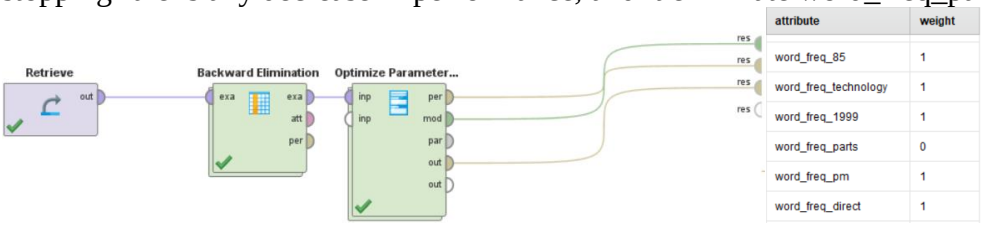
accuracy: 93.18% +/- 0.92% (micro average: 93.18%)

	true 1	true 0	class precision
pred. 1	1668	169	90.80%
pred. 0	145	2619	94.75%
class recall	92.00%	93.94%	

From here, we see that removing correlated terms slightly increase the accuracy.

   b. Backward Selection

      Here, we still using the same baseline model, but using backward selection, the stopping rule is any decrease in performance, and it eliminate word_freq_parts.



attribute	weight
word_freq_85	1
word_freq_technology	1
word_freq_1999	1
word_freq_parts	0
word_freq_pm	1
word_freq_direct	1

We can see from below that the performance slightly better than previous two.

accuracy: 93.28% +/- 0.29% (micro average: 93.28%)

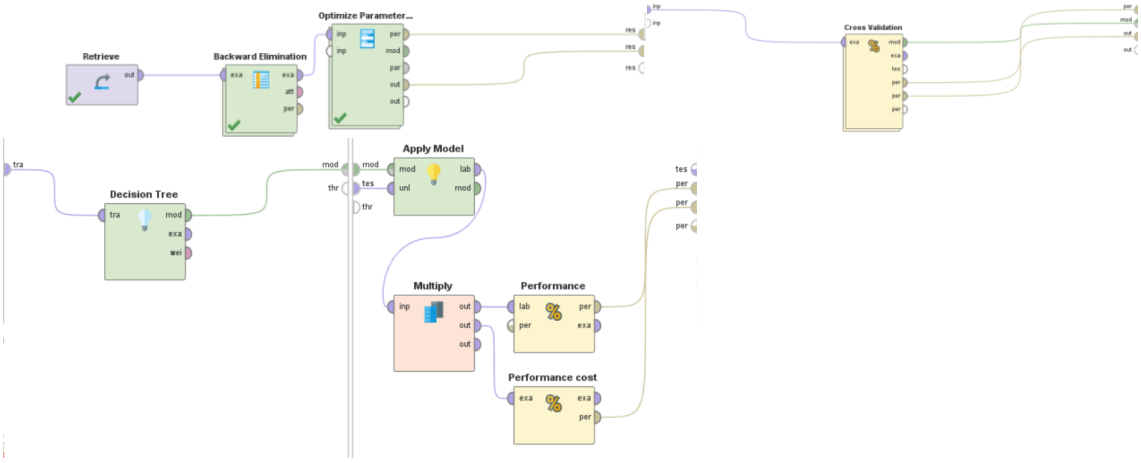
	true 1	true 0	class precision
pred. 1	1657	153	91.55%
pred. 0	156	2635	94.41%
class recall	91.40%	94.51%	

5. Model Selection
   a. Decision Tree

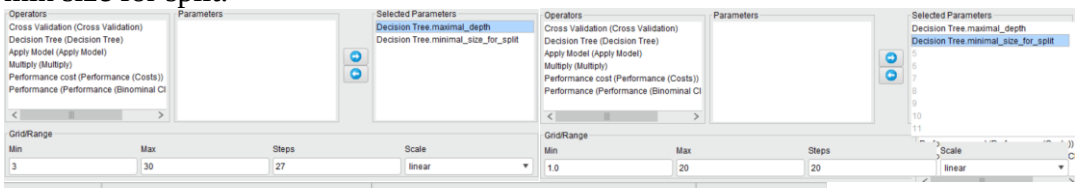
      Based on our knowledge from before, we first tried backward elimination and grid search on Decision tree.

      We previously already found that information gain is working better than other criterion in section 3, so we used info gain.

      Also, we found max depth and min split are the two binding parameters, so we do our grid search on those two.



The first model optimizing accuracy reported a 93.33% accuracy, with 14 depth and 1 min size for split.



iteration	Decision Tree.maximal_depth	Decision Tree.minimal_size_for_split	accuracy ↓
12	14	1	0.933
361	27	13	0.931
336	30	12	0.931

It gives AUC at 95.7%, Precision is at 94.54%, recall at 93.54%, and f measure is at 94.03%

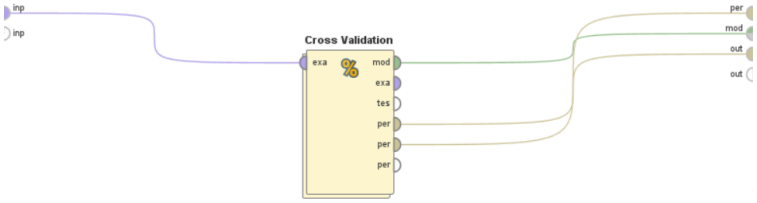
	true 1	true 0	class precision
pred. 1	1662	180	90.23%
pred. 0	151	2608	94.53%
class recall	91.67%	93.54%	

precision: 94.54% +/- 0.79% (micro average: 94.53%) (positive class: 0)

recall: 93.54% +/- 1.29% (micro average: 93.54%) (positive class: 0)

f_measure: 94.03% +/- 0.45% (micro average: 94.03%) (positive class: 0)

For the second model, we are optimizing average misclassification cost instead, and reported the lowest cost of 0.284 with 8 depth and 23 min split.



iteration	Decision Tree.maximal_depth	Decision Tree.minimal_size_for_split	Misclassificationcosts ↑
622	8	23	0.284
8	10	1	0.290
370	8	14	0.296

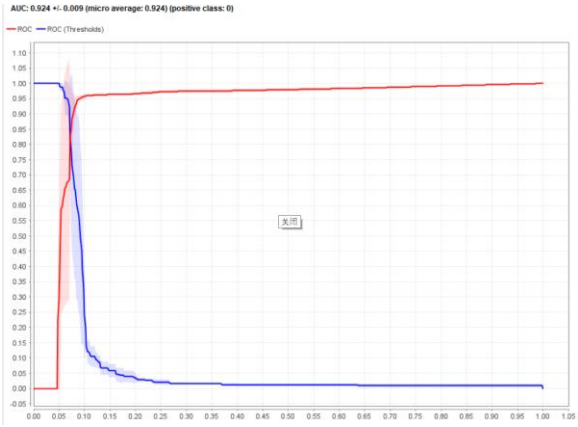
It gives AUC at 92.4%, Precision is at 94.55%, recall at 94.51%, and f measure is at 94.53%

	true 1	true 0	class precision
pred. 1	1661	153	91.57%
pred. 0	152	2635	94.55%
class recall	91.62%	94.51%	

precision: 94.55% +/- 0.72% (micro average: 94.55%) (positive class: 0)

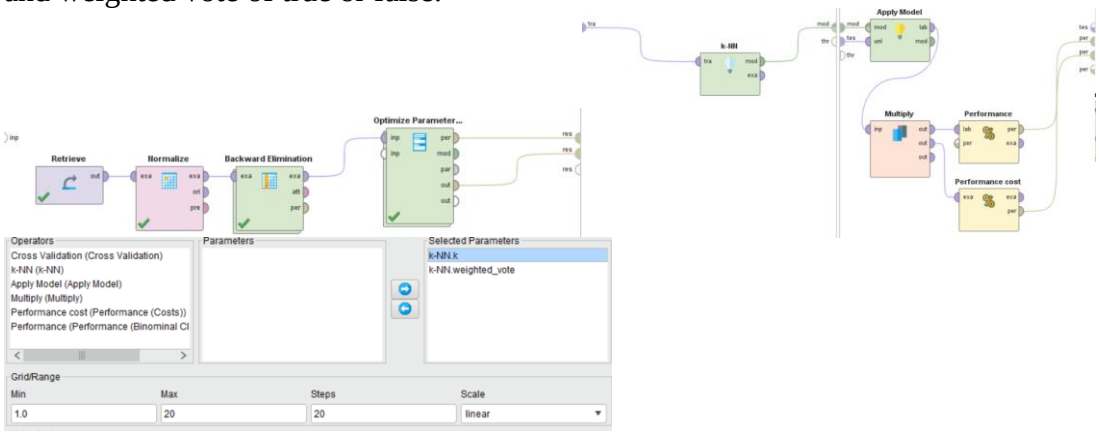
recall: 94.51% +/- 1.23% (micro average: 94.51%) (positive class: 0)

f_measure: 94.53% +/- 0.58% (micro average: 94.53%) (positive class: 0)



b. KNN

Here, we change the model inside the grid search to KNN, and normalize the data since KNN is sensitive to scales. For parameters, we used number of k from 1 to 20, and weighted vote of true or false.



The first model optimizing accuracy reported a 92.3% accuracy, with k of 5 and yes for weighted vote, two features are removed by the backward elimination.

iteration	k-NN.k	k-NN.weighted_vote	acc... ↓
4	4	true	0.923
8	3	false	0.917
3	3	true	0.917
5	5	true	0.916

It gives AUC at 96.3%, Precision is at 93.86%, recall at 93.47%, and f measure is at 93.65%

	true 1	true 0	class precision
pred. 1	1642	182	90.02%
pred. 0	171	2606	93.84%
class recall	90.57%	93.47%	

precision: 93.86% +/- 0.83% (micro average: 93.84%) (positive class: 0)

recall: 93.47% +/- 1.58% (micro average: 93.47%) (positive class: 0)

f_measure: 93.65% +/- 0.46% (micro average: 93.66%) (positive class: 0)

For the second model, we are optimizing average misclassification cost instead, and reported the lowest cost of 0.372 with k of 20 and yes for weighted vote.

iteration	k-NN.k	k-NN.weighted_vote	Mis... ↑
16	20	true	0.372
9	12	true	0.381
31	13	false	0.390
12	16	true	0.393
7	10	true	0.400

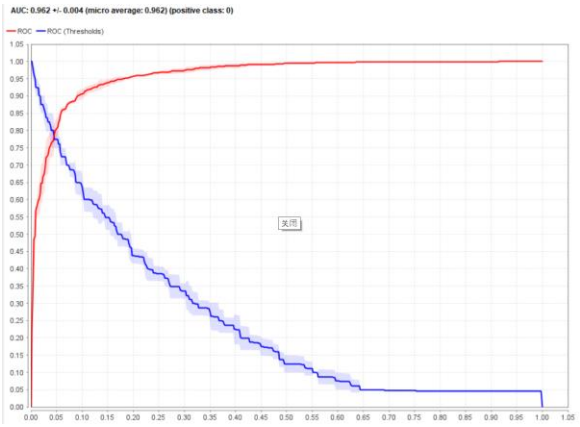
It gives AUC at 96.2%, Precision is at 89.82%, recall at 94.94%, and f measure is at 92.31%

	true 1	true 0	class precision
pred. 1	1513	141	91.48%
pred. 0	300	2647	89.82%
class recall	83.45%	94.94%	

precision: 89.82% +/- 0.77% (micro average: 89.82%) (positive class: 0)

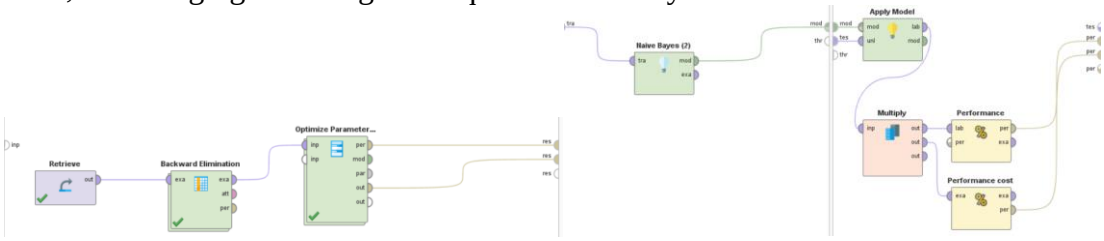
recall: 94.94% +/- 0.49% (micro average: 94.94%) (positive class: 0)

f_measure: 92.31% +/- 0.47% (micro average: 92.31%) (positive class: 0)



c. Naïve Bayes

Here, we changing modeling technique to Naïve Bayes with backward elimination.



The first model optimizing accuracy reported an 82.8% accuracy. For the second model, we are optimizing average misclassification cost and reported the lowest cost of 1.562. Naïve Bayes doesn't seems to be performing so well on this problem.

iteration	Naive Bayes (2).laplace_correction	accuracy ↓
2	false	0.828
1	true	0.826

iteration	Naive B...	Misclassificationc... ↑
2	false	1.567
1	true	1.572

It gives AUC at 93.3%, Precision is at 95.78%, recall at 74.89%, and f measure is at 84.25%

	true 1	true 0	class precision
pred. 1	1721	700	71.09%
pred. 0	92	2088	95.78%
class recall	94.93%	74.89%	

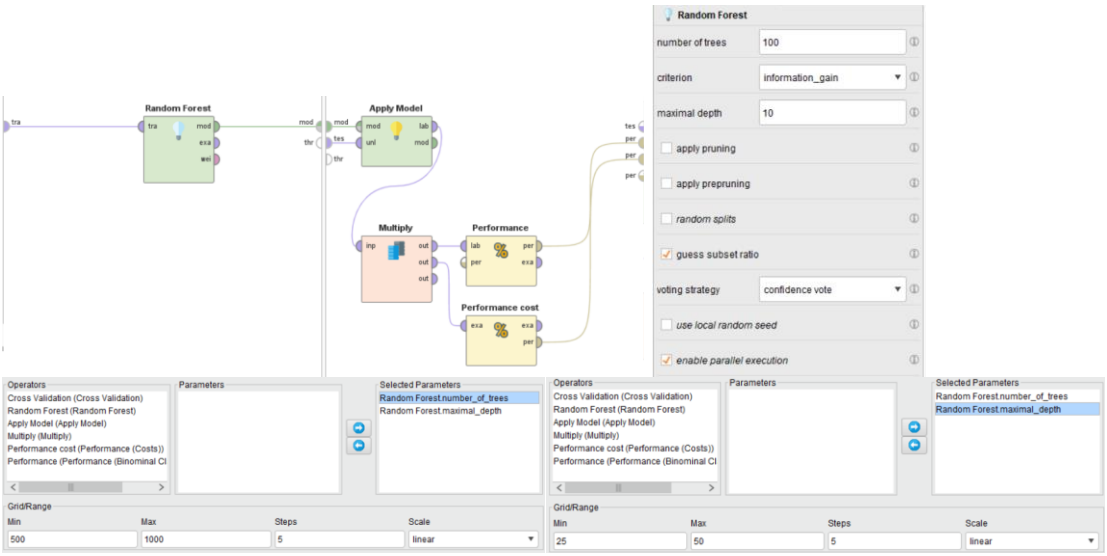
precision: 95.78% +/- 0.31% (micro average: 95.78%) (positive class: 0)

recall: 74.89% +/- 1.35% (micro average: 74.89%) (positive class: 0)

f_measure: 84.05% +/- 0.79% (micro average: 84.06%) (positive class: 0)

d. Random Forest

Here, we change the model inside the grid search to RandomForest, and for parameters, we used max depth from 25 to 50, and number of trees from 500 to 1000. According to knowledge from previous, we use information gain for each tree.



The random forest model report a best accuracy at 95.5% with multiply attribute combinition.

iteration	Random Forest.number_of_trees	Random Forest.maximal_depth	acc... ↓
13	500	35	0.955
11	900	30	0.955
26	600	45	0.955
24	1000	40	0.955
5	900	25	0.955
23	900	40	0.955
32	600	50	0.955
4	800	25	0.955
29	900	45	0.955
6	1000	25	0.954

Based on that result, we tried to min average misclassification cost and reported a lowest cost at 40 depth with 900 trees.

iteration	Random Forest.maximal_depth	Misclassificationcosts
3	40	0.195
4	45	0.202
2	35	0.213
1	30	0.207
5	50	0.205

Here, we further explore with other parameters, such as pruning and prepruning. After pruning, The best score we achieved is still at 95.5%. We further report the

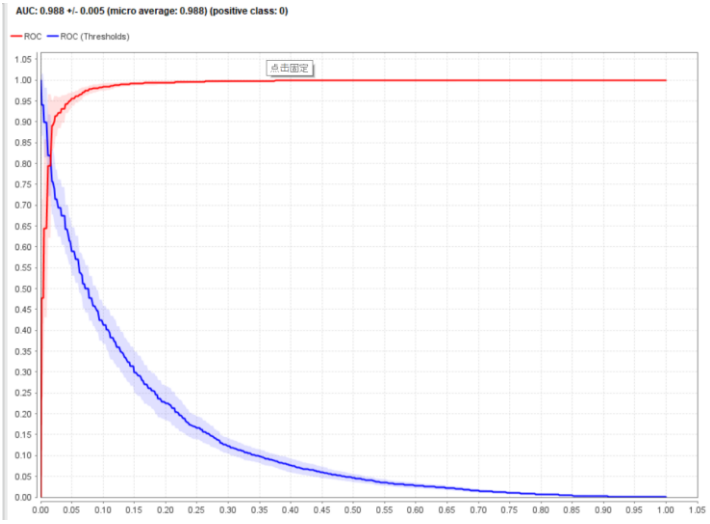
other scoring, where AUC is at 98.8%, Precision is at 95.2%, recall at 97.31%, and f measure is at 96.42%

	true 1	true 0	class precision
pred. 1	1676	75	95.72%
pred. 0	137	2713	95.19%
class recall	92.44%	97.31%	

precision: 95.20% +/- 0.75% (micro average: 95.19%) (positive class: 0)

recall: 97.31% +/- 1.19% (micro average: 97.31%) (positive class: 0)

f_measure: 96.24% +/- 0.62% (micro average: 96.24%) (positive class: 0)



After pruning, with confidence at 0.000 and minimal gain at 0.04, we report a lowest average misclassification cost at 0.163.

iteration	Random Forest.confidence	Random Forest.minimal_gain	Misclassificationcosts ↑
7	0.000	0.040	0.163
8	0.050	0.040	0.165
13	0.000	0.070	0.169
14	0.050	0.070	0.169
10	0.150	0.040	0.170
18	0.250	0.070	0.172
12	0.250	0.040	0.174
24	0.250	0.100	0.176
17	0.200	0.070	0.178
11	0.200	0.040	0.178
16	0.150	0.070	0.179
20	0.050	0.100	0.181
9	0.100	0.040	0.183
22	0.150	0.100	0.186
15	0.100	0.070	0.187
21	0.100	0.100	0.187
19	0.000	0.100	0.188
23	0.200	0.100	0.189
26	0.050	0.130	0.191

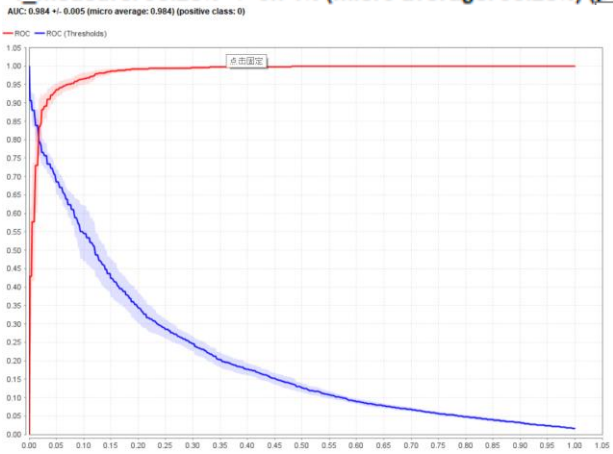
We further report the scoring of this best model, where AUC is at 98.4%, F measure is at 95.28%, precision at 92.93% and recall at 97.78%.

	true 1	true 0	class precision
pred. 1	1605	62	96.28%
pred. 0	208	2726	92.91%
class recall	88.53%	97.78%	

precision: 92.93% +/- 1.30% (micro average: 92.91%) (positive class: 0)

recall: 97.78% +/- 1.18% (micro average: 97.78%) (positive class: 0)

f_measure: 95.28% +/- 0.74% (micro average: 95.28%) (positive class: 0)



6. Best Accuracy

The most accurate model we found is using random forest with 900 trees, 40 depth, 1 min sample leaf. The optimized accuracy is at 95.5%. There are few produce same accuracy, but 900 trees and 40 depth have been found most stable producing this result.

iteration	Random Forest.number_of_trees	Random Forest.maximal_depth	acc... ↓
13	500	35	0.955
11	900	30	0.955
26	600	45	0.955
24	1000	40	0.955
5	900	25	0.955
23	900	40	0.955

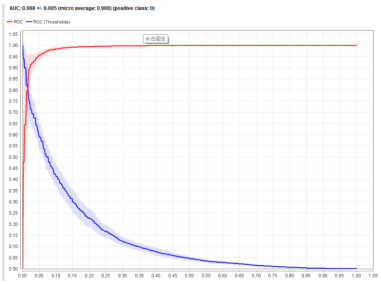
It also give best AUC at 98.8%, precision at 95.2%, recall at 97.31, f score at 96.24%, which are overall the best among all the classifers.

	true 1	true 0	class precision
pred. 1	1676	75	95.72%
pred. 0	137	2713	95.19%
class recall	92.44%	97.31%	

precision: 95.20% +/- 0.75% (micro average: 95.19%) (positive class: 0)

recall: 97.31% +/- 1.19% (micro average: 97.31%) (positive class: 0)

f_measure: 96.24% +/- 0.62% (micro average: 96.24%) (positive class: 0)



7. Best Cost-sensitive

   The best cost-sensitive model we got is using random forest, with 900 trees, 40 depth, 1 min sample leaf, 0.000 confidence and 0.04 minimal gain. The optimized average misclassification cost is reported at 0.163.

iteration	Random Forest.confidence	Random Forest.minimal_gain	Misclassificationcosts ↑
7	0.000	0.040	0.163
8	0.050	0.040	0.165
13	0.000	0.070	0.169

   Compare to the best accuracy model, we can see that the cost-sensitive model has less false positive cases of 62, higher recall, but lower accuracy/precision/f-measure/AUC. It gives best economic decision, but sacrifices some accuracy as a trade off.

	true 1	true 0	class precision
pred. 1	1605	62	96.28%
pred. 0	208	2726	92.91%
class recall	88.53%	97.78%	

precision: 92.93% +/- 1.30% (micro average: 92.91%) (positive class: 0)

recall: 97.78% +/- 1.18% (micro average: 97.78%) (positive class: 0)

f_measure: 95.28% +/- 0.74% (micro average: 95.28%) (positive class: 0)