(Kaggle Username: dlim057)

Alternative representations for attributes

I am using the program Weka to create my models and edit the data set. To start off, I removed the ID attribute as this has no relationship to the class attribute and will only create more noise for the models. I also changed the class attribute, 'Y' from a numeric to a nominal value as it has only 2 values, 1 and 0. Without this change I would not be able to create a model for the data set. Next, I looked into other attributes I should change from a numeric to nominal value. I concluded that X2(Region), X3(Gender), X4(Education), X5(Marital Status), and X7 to X12(History of past payment) should be changed to a nominal value.

With this small change, I decided to test out a J48 algorithm decision model and 10-fold cross-validated it with the training data set. The result (See Appendix CV1) turned out better than expected with an accuracy of 0.817. However, the True Positive rate for defaulting on the credit card payments was only 0.370. I ran the model on the testing set and uploaded to Kaggle which got a score of 0.82133.

Outliers and Standardisation

I moved on to identifying and removing outliers that might affect the accuracy of the model. I used the 'InterquartileRange' filter to find and 'RemoveWithValues' to delete the outliers on Weka. Weka identified and removed exactly 2862 entries from the training data set. I ran a cross-validation on the new model with the same J48 algorithm. (See Appendix CV2) Although the resulting overall accuracy has decreased, the TP rate for defaulting on the credit card payment did increase from 0.370 to 0.387. The lower overall accuracy could be the result of removing the outliers that had been inflating the previous accuracy, as a large proportion of the training set's class attribute is set as 0. Therefore, seeing an improvement in the true positive rate for defaulting is an improvement to the overall model. I then standardised the data set using the 'Standardize' filter so that all the good data that is left after removing the outliers were standardised.

Removing attributes

Next, I removed certain attributes from the data set by looking at their relationship with the class attribute. Just from a visual standpoint, I can see that X2(Region), X3(Gender), X5(Marital status) X6(Age) have a very weak relationship with the class attribute. The attribute X5 is shown (See Appendix A1) as an example. Also, upon further inspection, I could see that the attributes X19 to X24 were also weakly related to the class attribute, and therefore was removed from the data set.

Ensembles and Classifiers

Next step was to explore different ensembles and classifiers to be used in my models because so far, the only algorithm I've used was the J48 algorithm. For every new model, I will be cross validating 3-fold due to the building time of some models. To make sure I do not overfit the model for the training set, I will be running it on the testing set and uploading it to Kaggle for every new model.

The first algorithm I looked at is a simple Naïve Bayes algorithm. The result (See Appendix CV3) did not go over the accuracy of the J48 algorithm as expected. However, surprisingly it had a much higher TP rate for defaulting than any other previous cross-validated result with 0.438. It still scored less than the current highest score of 0.82133 on Kaggle with 0.80033.

Bagging

The first ensemble I experimented with was different bagging algorithms starting with the Randomforest algorithm. I set the bag size at 50% of the data set and set 100 iterations for the algorithm. The cross-validation (See Appendix CV4) for the algorithm turned out to be less accurate than the J48 decision model made with the unedited data set, with an accuracy of 0.801. It also had a lower TP rate for defaulting. Surprisingly on Kaggle it scored a 0.82233 which was higher than the previous score of 0.82133.

I tried a new bagging algorithm with J48 as the classifier, setting the bag size at 50% and 30 iterations. It did better than the random forest algorithm with an accuracy of 0.806 (See Appendix CV5). On Kaggle it scored a 0.82400, which was another improvement from 0.82233.

I changed the classifier to REPTree, keeping the bag size at 50% with 30 iterations. The cross-validation result (See Appendix CV6) was similar to the result of using J48 as the classifier with an accuracy of 0.808. On Kaggle, this model scored a 0.82400 which was the same score as the J48 bagging algorithm.

The final bagging classifier I used was the LMT classifier with 50% bag size and 30 iterations. This model took an extremely long time to build (559.39 seconds) and the cross-validation result (See Appendix CV7) came out like the previous bagging algorithms. However, on Kaggle it scored higher with a score of 0.82466. However, due to the time taken to build the model, experimenting with this classifier seems impractical compared to a classifier like REPTree or J48.

Stacking

Next, I looked into stacking with the use of 4 different classifiers and a meta classifier. The classifiers I used are RandomForest(Bag size 50%, 100 iterations), Naïve Bayes, Decision stump, and Simple logistic. The meta classifier was a bagging algorithm with J48 as the classifer with 50% bag size and 30 iterations. I decided to use this meta classifier as it is one of the best performing models so far. The cross-validation result (See Appendix CV8) was slightly better than any other algorithms tested so far but the score on Kaggle was 0.82166 which was not an improvement. The time taken to build the model was also long with 268 seconds.

I tried a different stacking algorithm with different classifiers. I used J48, REPTree, Randomforest(50% bag size, 100 iterations), and SimpleLogistic as the classifiers and Bagging with J48(50% bag size, 30 iterations) as the meta classifier. This model look less time to build with 170 seconds. The cross-validation result (See Appendix CV9) was slightly better than the previous stacking algorithm with a higher TP rate for defaulting. On Kaggle it scored a 0.82133.

Boosting

Finally, I tried boosting with the AdaBoostM1 algorithm on Weka. The first classifier I used was a SimpleLogistic classifier with 10 iterations. The 3-fold cross-validation result (See Appendix CV9) was comparable to the stacking result, but not an improvement. On Kaggle the predictions using this model scored a 0.82200.

I changed up the classifier to the J48 classifier and increased the number of iterations to 30. Although the overall accuracy of the cross-validation (See Appendix CV10) was low, the TP rate for defaulting was higher than any recent models tested except for the Naïve Bayes algorithm. This model scored a 0.80133, one of the lowest scores so far.

Instead of just using the J48 as a classifier, I decided to combine bagging and boosting by using the J48 algorithm as a classifier for bagging and using that as the classifier for AdaBoostM1. I set the bag size and number of iterations as 50% and 30 respectively and changed it back down to 10 iterations for boosting. The cross-validation (See Appendix CV11) returned another high TP rate for defaulting and overall accuracy was low. On Kaggle this model scored a 0.80066. I concluded that boosting was not a very good ensemble to create models with for this data set.

Revisiting Bagging

Since bagging seems like the best ensemble to work with, I revisited it by changing the values for different parameters. J48 as the classifier was one of the highest scoring models so far on Kaggle and time wise it is the most viable. Since the original parameters were 50% of 30 iterations, I reduced both down to 30% of 20 iterations to make sure I was not overfitting the training data. The cross-validation result (See Appendix CV12) came back more or less the same as the previous version. However, on Kaggle it scored lower with 0.82100.

Judging by this result, I increased the number of iterations to 100 and changed the bag size to the original 50%. Again, the cross-validation result (See Appendix CV13) came back very similar to previous versions. On Kaggle it scored a 0.82333. Due to the very similar cross-validation results as I am only changing the parameters of the same algorithm, from this point forwards I will only be discussing the performance on Kaggle.

This time I increased the bag size to 80% and changed the number of iterations at 30. On Kaggle, this model scored 0.82300. This leads me to believe that the number of iterations have a larger impact on the accuracy of the model.

To test this, I decided to increase the number of iterations to 300 and take the bag size back down to 50%. On Kaggle, this model a 0.82400. The result shows that maybe changing the number of iterations is not as impactful as I had previously thought.

Next, I tried something new by making a bagging J48 algorithm as the classifier for another bagging algorithm. I used the same 50%, 30 iterations for both bagging algorithms. This model scored a 0.82366 on Kaggle.

I moved on to testing out the REPTree as a classifier for bagging again. I changed the bag size to 40% and the iterations to 200. This model saw the same score as the original Bagging algorithm with an LMT classifier with 0.82466.

From my analysis having a bag size of 40% with 200 iterations is the best values for parameters, and so I used the LMT classifier again with these settings. However, this theory was proven wrong because this model only scored a 0.82300 on Kaggle.

Conclusion

In conclusion, bagging seems to be the best ensemble for data sets for defaulting on credit card payments. Boosting and stacking did not have the same accuracy on the testing data as bagging did even with different classifiers and parameters. Out of the classifiers, LMT and REPTree showed the most promise with the highest Kaggle scores even with similar cross-validation results compared to other classifiers. My 2 final models I will be submitting is the Bagging(REPTree, 40% bag size and 200 iterations) and the Bagging(LMT, 50% bag size and 30 iterations) algorithms. These two models both scored a 0.82466 on the leaderboards. I believe from experimenting with various combinations of ensembles and classifiers, this is the best model I have created for the testing data set.

Appendix

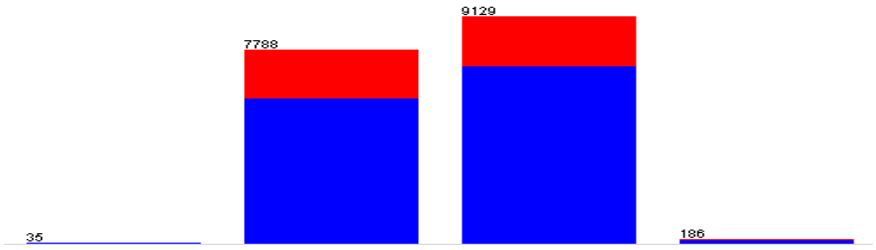
CV1

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.945	0.630	0.840	0.945	0.889	0.395	0.676	0.833	0
	0.370	0.055	0.655	0.370	0.473	0.395	0.676	0.440	1
Weighted Avg.	0.817	0.503	0.799	0.817	0.797	0.395	0.676	0.746	

CV2

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.935	0.613	0.832	0.935	0.881	0.393	0.681	0.826	0
	0.387	0.065	0.648	0.387	0.484	0.393	0.681	0.445	1
Weighted Avg.	0.806	0.484	0.789	0.806	0.788	0.393	0.681	0.737	

A1



CV3

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.895	0.562	0.838	0.895	0.865	0.364	0.747	0.882	0
	0.438	0.105	0.561	0.438	0.492	0.364	0.747	0.520	1
Weighted Avg.	0.787	0.455	0.773	0.787	0.778	0.364	0.747	0.797	

CV4

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.936	0.640	0.826	0.936	0.878	0.370	0.744	0.879	0
	0.360	0.064	0.635	0.360	0.460	0.370	0.744	0.524	1
Weighted Avg.	0.801	0.504	0.781	0.801	0.780	0.370	0.744	0.796	

CV5

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.941	0.631	0.829	0.941	0.881	0.388	0.748	0.877	0
	0.369	0.059	0.657	0.369	0.472	0.388	0.748	0.543	1
Weighted Avg.	0.806	0.497	0.788	0.806	0.785	0.388	0.748	0.799	

CV6

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.946	0.639	0.828	0.946	0.883	0.391	0.763	0.892	0
	0.361	0.054	0.672	0.361	0.469	0.391	0.763	0.553	1
Weighted Avg.	0.808	0.502	0.791	0.808	0.786	0.391	0.763	0.812	

CV7

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.945	0.631	0.830	0.945	0.884	0.398	0.762	0.891	0
	0.369	0.055	0.674	0.369	0.477	0.398	0.762	0.557	1
Weighted Avg.	0.810	0.496	0.793	0.810	0.788	0.398	0.762	0.812	

CV8

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.946	0.639	0.828	0.946	0.883	0.393	0.753	0.879	0
	0.361	0.054	0.674	0.361	0.470	0.393	0.753	0.545	1
Weighted Avg.	0.809	0.501	0.792	0.809	0.786	0.393	0.753	0.800	

CV9

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.945	0.628	0.830	0.945	0.884	0.400	0.752	0.879	0
	0.372	0.055	0.675	0.372	0.479	0.400	0.752	0.546	1
Weighted Avg.	0.810	0.493	0.794	0.810	0.789	0.400	0.752	0.801	

CV10

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.948	0.641	0.828	0.948	0.884	0.395	0.733	0.873	0
	0.359	0.052	0.681	0.359	0.470	0.395	0.733	0.491	1
Weighted Avg.	0.810	0.502	0.793	0.810	0.787	0.395	0.733	0.783	

CV11

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.900	0.624	0.824	0.900	0.861	0.316	0.692	0.850	0
	0.376	0.100	0.537	0.376	0.443	0.316	0.692	0.464	1
Weighted Avg.	0.777	0.500	0.757	0.777	0.762	0.316	0.692	0.759	

CV12

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.890	0.616	0.824	0.890	0.856	0.306	0.691	0.854	0
	0.384	0.110	0.518	0.384	0.441	0.306	0.691	0.455	1
Weighted Avg.	0.771	0.497	0.752	0.771	0.758	0.306	0.691	0.760	

CV13

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.944	0.635	0.829	0.944	0.883	0.393	0.740	0.870	0
	0.365	0.056	0.669	0.365	0.473	0.393	0.740	0.536	1
Weighted Avg.	0.808	0.498	0.791	0.808	0.786	0.393	0.740	0.792	

CV14

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.942	0.637	0.828	0.942	0.881	0.386	0.753	0.881	0
	0.363	0.058	0.660	0.363	0.469	0.386	0.753	0.548	1
Weighted Avg.	0.806	0.500	0.788	0.806	0.784	0.386	0.753	0.803	