CS 6923 Machine Learning Spring 2019
Final Project Report

Name: Aline Bessa
NetID: ab5149

Name: Rao Li
NetID: rl3427

PART I: Preprocessing (No more than two pages for this part)

1. How does your program handle missing values? And why?

There are five columns with missing values in the given training set: *weight, house, player_code, move_specialty,* and *gender*. We applied the following methods to handle these missing values.

- Drop feature

Since *weight* has about 97% missing values, it is hard to apply any technique to replace the missing values. So we should consider this case as missing information and thus drop this feature.

- Assign missing values to new category

Because there is still much unknown information in *house*, *player_code* and *move_specialty*, and they are nominal values, we created a new category called 'U' and assigned these missing values to it to prevent the loss of more information.

- Drop rows

In the *gender* feature, there are a few rows with "Unknown/Invalid". Because we have about 100,000 examples in the training set and only about three of them are missing, we dropped these rows with "Unknown/Invalid", not harming the result and facilitating the use of binary encoding on the *gender* feature latter.

2. If your program converts numeric features to categorical features, or categorical features to numeric features, describe how it does it.

According to the dataset information given, all of the categorical values are nominal, so there is no order among them. Consequently, we applied binary encoding and one-hot encoding instead of label encoding to prevent the inclusion of incorrect information.

- Binary encoding

Binary encoding can only be used for features with fewer than three nominal values. In this project, all columns of tactics (except for *finbourgh_flick* and *double_eight_loop*, which will be discussed in the next part) are encoded with binary encoding after the reduction of feature values (details explained in the next part). Also, *gender* (after removing Unknown type), *snitch_caught*, *move_specialty* (after reducing feature values), *change* and the target are encoded using binary encoding.

- One-hot encoding

One-hot encoding is used for encoding categorical features with more than two nominal values. We applied one-hot encoding on *house*, *foul_type_id*, *game_move_id*, *penalty_id*, *player_code*, *player_type*, *snitchnip* and *stooging*.

3. Describe any feature selection, combination or creation, and any feature values combination performed by your program and the reasons for doing so.

- Correlation-based feature selection

By just observing the training dataset, we decided to remove *player_id* because this feature only stands for the player's identity, which cannot be correlated to the target. Also, we removed *finbourgh_flick* and *double_eight_loop* because each of these two features has the same value, which could be considered a missing information problem. We also output the correlation matrix after cleaning the training data. By observing the matrix, we noticed that there is no significantly high correlation between any of the feature pairs (the higher correlations are about 70-80%, but we expected more). This is surprising, as we derived new features from the original features (this will be discussed later). So no feature was dropped at this step because our goal is to prevent the dismissal of useful information.

- Greedy feature selection

After gathering results for different models, we came back to the preprocessing part to verify whether some features were adding noise to the problem, hampering the models' effectiveness. Ideally, we should execute the models for all possible feature combinations, but this approach does not scale. We then greedily selected features --- one by one in order, starting from *gender* --- and verified if our best model (discussed in Part II) would perform better. If it did not, we dropped the feature. We noticed that although some features did not have to be added according to this greedy criterion, they were not hampering the performance, and keeping all of them still gave us better results. So no feature was dropped at this step.

- Feature combination and creation

This step aims to (i) combine correlated features based on dataset information, and (ii) create new ones which have more information and might be more related to the target, helping with prediction. Three new features were created:

1. *num_game_not_participate* - this feature is derived from the sum of *num_games_satout*, *num_games_injured* and *num_games_notpartof*.
2. *num_tactics_change* - this feature is created from tactics features. There are four nominal values among tactics: *Steady*, *Up*, *Down*, *No*. *Up* and *Down* indicate that the average usage of the tactic does not change. So *num_tactics_change* can be calculated by counting 'Up' and 'Down' for each player.
3. *num_total_tactics* - this feature is also derived from tactics features. Different from above, *num_total_tactics* was calculated by counting *Steady*, *Up* and *Down*, which only focus on number of total tactics each player used instead of change.

- Feature values combination

Feature values combination aims to reduce the number of nominal values in features. This can be helpful when performing one-hot encoding, preventing newly derived features from having most of the values being the same. For example, a particular nominal value could be very rare in the feature. If we apply one hot encoding on this feature, we would get a newly derived feature with most values being 0, and only few entries being 1 --- this may not help the prediction task. So the reason to combine feature values is similar to the one for combining features: gathering information together and deriving more useful new features. We did feature values combination on the following features:

1. *snitchnip* -">200" and ">300" are combined into one feature value named *hig*h
2. *stooging* - ">7" and ">8" are combined into one feature value named *high*
3. *move_specialty* - we encoded all *move_specialty* values, regardless of their id, as 1 and missing type 'U' as 0.
4. *tactics* - *Steady*, *Up*, *Down* are encoded as 1, and *No* is encoded as 0.

4. Describe other preprocessing used in your program(e.g. centralizing, normalization)

In order to represent all numerical values in a same scale, we used log transformation and standardization. We need to scale the data because the value ranges vary across features, and higher values can have larger weights, affecting the models unfairly. So before performing standardization, log transformation was applied on features whose values have high skew (*num_games_satout, num_games_injured* and *num_games_notpartof*), thus stretching low values and compressing the high ones, making the distribution of values more similar to a normal distribution. The last step of preprocessing data is to standardize all numerical data to scale them to a same value range. The formula we used to standardize a value in a distribution is: (value-mean)/standard-deviation.

PART II: Classification (No more than two pages for each model in this part)
Model One:

1. **Supervised learning method used in this model is**

Logistic Regression

2. **Why did you choose this supervised learning method?**

We picked it as a baseline because:

- it is very efficient and scales well as the number of features grows
- its outputs are easy to interpret with scikit-learn

3. **Describe the method you used to evaluate this method.**

As the classes are very imbalanced, we decided to evaluate this method by:

- Performing 5-fold cross validation and, for each folder, computing the related **F1 score** per class. We then generated the mean F1 score for both classes in each folder, and in the end took the mean for the 5 folders. In other words, we used the *mean F1 score* **(F)** as our metric.

4. **Describe the process of experimenting different parameter settings or associated techniques.**

We experimented with different parameter settings in a greedy fashion, fixing one parameter value at a time. We used scikit-learn's implementation.

○ Standard result:

- Without tuning, the mean F1 score we obtain with 5-fold cross-validation is **F = 0.475806621975**.

○ Sampling the training data:

- Parameter values:
  - *Oversampling* the minority class (SMOTE) and *undersampling* the majority class
- Performance of different values:
  - With *oversampling*, we boosted the number of examples for class YES and ended up with **F = 0.525442670882**.
  - With *undersampling*, we reduced the number of NO examples randomly, also reaching balance. In this case, F = 0.522378366653.
- Analysis:
  - The mean score obtained with *oversampling* was significantly better than the standard one, so we used ***oversampling*** in all our subsequent experiments. Important: *we only oversample the examples that are used in the training folds.* The test fold is always left intact, thus remaining imbalanced.

○ C (Inverse of regularization strength):

- Parameter values:
  - 0.001, 0.01, 0.1, 1, 10, 100, 1000, and 10000
- Performance of different values:
  - Respectively, C = 0.001, C = 0.01, C = 0.1, C = 1, C = 10, C = 100, C = 1000, and C = 10000 led to F = 0.522619397466, **F = 0.525880380014**, F = 0.525521476924, F = 0.525442670882, F = 0.525287847474, F = 0.525233303313, F = 0.525550509832, F = 0.525233140099 and F = 0.52525391557.
- Analysis:
  - We started this experiment with a high amount of regularization (increasing the Bias^2 of the model), and progressed towards a model with higher variance. The best result is for **C = 0.01**, i.e., this model benefits from strong regularization (reasonable, as the number of features is relatively high). Note, however, that the results change very little, i.e, this model is robust with respect to regularization.

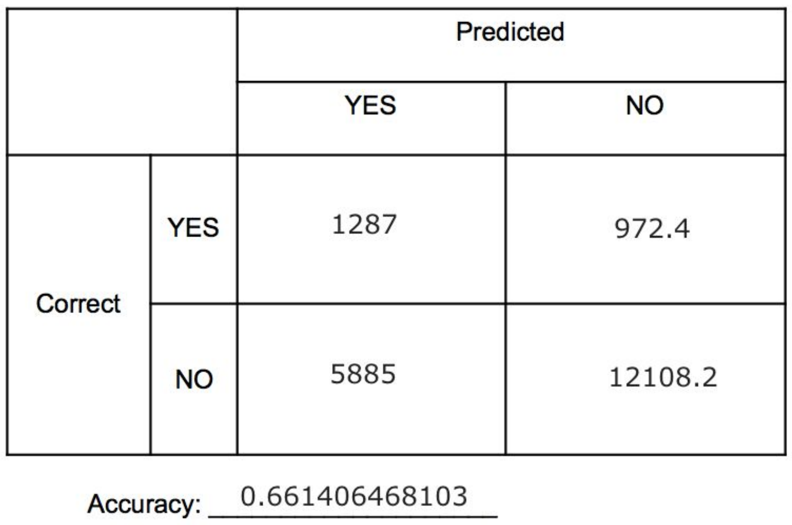
○ class_weight (relevant when the data is too imbalanced):
- Parameter values:
  - *None* and *balanced*
- Performance of different values:
  - **F = 0.525880380014** for both values.
- Analysis:
  - Tweaking class_weight did not change the value of F, maybe because we had already artificially balanced the training data with oversampling. We then fixed **class_weight = None.**

○ penalty (used to specify the norm used in the regularization):
- Parameter values:
  - *L1* and *L2*
- Performance of different values:
  - Respectively, we got F = 0.525212207167 and **F = 0.525880380014** for penalty = L1 and penalty = L2.
- Analysis:
  - We fixed **penalty = L2** because its corresponding F is slightly higher.

○ solver (algorithm used for the optimization):
- Parameter values:
  - lbfgs, liblinear, sag
- Performance of different values:
  - Respectively, we got F = 0.526037165742, F = 0.525880380014, and **F = 0.526098499947** for lbfgs, liblinear and sag.
- Analysis:
  - We fixed **solver = sag** because it led to the highest F value. Note that changes in the parameters of the model *per se* did not change F substantially. The most significant impact was obtained by using **oversampling.**

5. Accuracy and Confusion matrix with most suitable parameters

In the table below, we use the average values for the 5 folds.



		Predicted	
		YES	NO
Correct	YES	1287	972.4
	NO	5885	12108.2

Accuracy: 0.661406468103

Model Two:

1. **Supervised learning method used in this model is**

Adaboost with Decision Trees as weak learners.

2. **Why did you choose this supervised learning method?**

We picked Adaboost because:

- it is very robust with respect to parameter changes
- i is an ensemble method, which theoretically makes it very powerful

3. **Describe the method you used to evaluate this method.**

For Adaboost, we used the same evaluation scheme we fixed for Logistic Regression. In fact, we fixed a random state for the cross validation and tested all models over the same dataset splits, making them directly comparable.

4. **Describe process of experimenting different parameter settings or associated techniques.**

We experimented with different parameter settings in a greedy fashion, fixing one parameter value at a time. We used scikit-learn's implementation.

○ Standard result:

- Without tuning, the mean F1 score we obtain with 5-fold cross-validation is **F = 0.482964752111**.

○ Sampling the training data:

- Parameter values:
  - *Oversampling* the minority class (SMOTE) and *undersampling* the majority class
- Performance of different values:
  - With *oversampling*, we boosted the number of examples for class YES and ended up with **F = 0.526827185398**.
  - With *undersampling*, we reduced the number of NO examples randomly, also reaching balance. In this case, F = 0.52361215995.
- Analysis:
  - The mean score obtained with ***oversampling*** was significantly better than the standard one, so we used oversampling in all our subsequent experiments.

○ Decision Tree's max_depth (Shallowness of the weak classifiers):

- Parameter values:
  - 1, 2, 4, and 8
- Performance of different values:
  - Respectively, max_depth = 1, max_depth = 2, max_depth = 4, and max_depth = 8 led to **F = 0.526827185398**, F = 0.515354437309, F = 0.512175883466, and F = 0.520901060524.
- Analysis:
  - For this particular setting, it looks like shallow classifiers perform better (we fix **max_depth = 1**). There is, however, an increase in F when max_depth = 8. We do not believe this is an overfitting issue because this result is a mean over cross validation folds, and did not carry experiments with larger max_depth values because the performance was degrading fast.

○ n_estimators (number of estimators at which boosting is terminated)

- Parameter values:
  - 10, 25, 50, 100, 200, 500
- Performance of different values:

- Respectively, n_estimators = 10, n_estimators = 25, n_estimators = 50, n_estimators = 100, n_estimators = 200, and n_estimators = 500 led to F = 0.510620669991, F = 0.52350181569, F = 0.526827185398, F = 0.526760528534, **F = 0.528356063423**, F = 0.527770678244.

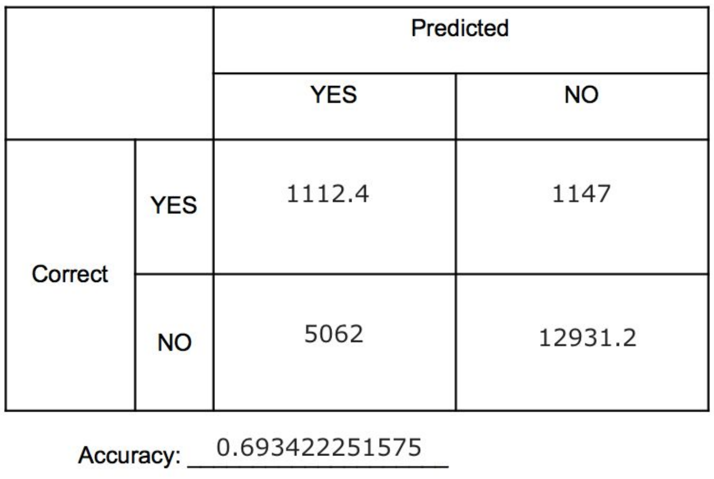
- Analysis:
  - The best mean F we obtained is linked to **n_estimators = 200**, indicating that Adaboost benefits from longer boosting pipelines. With n_estimators = 500 we obtain a slightly worse F, maybe because the boosting starts to overfit the weights given to the instances.

○ algorithm (boosting algorithm)

- Parameter values:
  - *SAMME* and *SAMME.R*
- Performance of different values:
  - Respectively, algorithm = SAMME and algorithm = SAMME.R led to **F = 0.534905655781** and F = 0.528690883968.
- Analysis:
  - Although we had read that SAMME.R usually leads to best results, besides converging faster, we got better values with **algorithm = SAMME**. As with Logistic Regression, **oversampling** was the change that brought the most impact to the value of F. Variations in Adaboost parameters did not affect the final F much.

5. Accuracy and Confusion matrix with most suitable parameters

In the table below, we use the average values for the 5 folds.



		Predicted	
		YES	NO
Correct	YES	1112.4	1147
	NO	5062	12931.2

Accuracy: 0.693422251575

Model Three:

2. **Supervised learning method used in this model is**

Random forest.

2. Why did you choose this supervised learning method?

We picked Random Forest because:

- it is very robust with respect to overfitting
- it also is an ensemble method, which is suitable for harder classification problems, such as those with class imbalance.

3. Describe the method you used to evaluate this method.

For Random Forest, we used the same evaluation scheme we fixed for Logistic Regression and Adaboost. All results were obtained over the same data division, and are thus directly comparable.

4. Describe process of experimenting different parameter settings or associated techniques.

We experimented with different parameter settings in a greedy fashion, fixing one parameter value at a time. We used scikit-learn's implementation.

○ Standard result:

- Without tuning, the mean F1 score we obtain with 5-fold cross-validation is **F = 0.491253452133**.

○ Sampling the training data:

- Parameter values:
  - *Oversampling* the minority class (SMOTE) and *undersampling* the majority class
- Performance of different values:
  - With *oversampling*, we boosted the number of examples for class YES and ended up with **F = 0.512173128578**.
  - With *undersampling*, we reduced the number of NO examples randomly, also reaching balance. In this case, F = 0.503710443379.
- Analysis:
  - The mean score obtained with ***oversampling*** was significantly better than the standard one, so we used oversampling in all our subsequent experiments.

○ n_estimators (Number of decision trees in the forest):

- Parameter values:
  - 5, 10, 25, 50, 75 and 100.
- Performance of different values:
  - Respectively, n_estimators = 5, n_estimators = 10, n_estimators = 25, n_estimators = 50, n_estimators = 75, and n_estimators = 100 led to **F = 0.535239173803**, F = 0.511685120141, F = 0.512022538305, F = 0.504260687469, F = 0.506834091925, and F = 0.503397580613.
- Analysis:
  - Surprisingly, the best result we obtained was with the smallest number of estimators (**n_estimators = 5**). It is possible that the results would change again, becoming better for larger values of n_estimators, such as 1000. Unfortunately, we did not have time to experiment with larger values due to hardware limitations. Alternatively, it is possible that fewer trees are better for this setting, capturing less noise from the large number of available features.

○ criterion (the function to measure the quality of a split)

- Parameter values:
  - *gini* and *entropy*

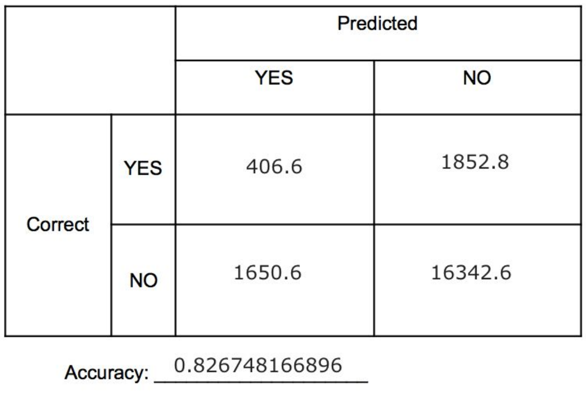
- Performance of different values:
  - Respectively, criterion = *gini* and criterion = *entropy* led to F = 0.533727903942 and F = 0.53400583093.
- Analysis:
  - The best mean F we obtained is linked to **criterion = *entropy***, i.e., in this problem, separating examples by computing the information gain is a bit better than calculating the Gini impurity. The results are, however, rather similar.

○ max_depth (the maximum depth for the trees)
- Parameter values:
  - 2, 8, 32, None
- Performance of different values:
  - Respectively, max_depth = 2, max_depth = 8, max_depth = 32, and max_depth = None led to F = 0.489388001105, F = 0.499387325396, **F = 0.540235603988**, and F = 0.534554116748.
- Analysis:
  - The tree depth is crucial for the F1 score in our problem, probably because there are many different variables that can be used to split the examples, and exploring them thoroughly leads to deeper trees. We fixed **max_depth = 32** because the result was the best, but also because we do not want trees that are much longer than that because they may overfit future test data.

○ max_features (the number of features considered when looking for the best split)
- Parameter values:
  - *auto* (sqrt(n_features)), *log2* (log2(n_features)), and *None* (n_features)
- Performance of different values:
  - Respectively, for max_features = *auto*, max_features = *log2,* and max_features = *None*, we obtained F = **0.540235603988**, F = 0.539212947883, and **F = 0.543479338507**.
- Analysis:
  - We fixed **max_features = *None*** because it led to best value. In practice, though, the performance is not good when so many features need to be analyzed for every split, so we would probably use max_features = *sqrt* more often, as the result does not seem to differ much. Different from Logistic Regression and Adaboost, the use of oversampling was not the only parametrization that had a significant impact: the choice of a suitable value for n_estimators was also important.

5. Accuracy and Confusion matrix with most suitable parameters

In the table below, we use the average values for the 5 folds.



		Predicted	
		YES	NO
Correct	YES	406.6	1852.8
	NO	1650.6	16342.6

Accuracy: 0.826748166896

PART III: Best Hypothesis (No more than two pages for this part)

1. Which model do you choose as final method?

Model number: 3.

Supervised learning method used in this model: Random Forest.

2. Reasons for choosing this model.

- Highest F1 score among studied options.
- *Significantly* higher accuracy when compared to the other models.
- It performs well, scaling better than Adaboost --- both for larger parameter values and for larger amounts of training data.
- It is easy to profile the leaves of its trees and understand how the classification is being made. In other words, it is a highly interpretable model.

3. What are the reasons do you think that make it has the best performance?

- It has a very high recall for class NO, i.e., the true negative rate is large. This has a positive consequence in the accuracy and does not affect the F1 score much. In fact, the F1 score is a bit better as well, in comparison with other models.
- We believe that tuning the number of estimators properly was important to balance the amount of variance, which had a very positive impact in the model. Even if the recall for class YES could be better, the model is not attempting to learn very complicated hypotheses to adjust for a minority of examples.