CS63 Spring 2016
Exploring Classification Techniques on Animal Shelter Intake Data

Tuan Nguyen (tnguyen4) and Tyrone Clay (tclay1)

5/9/2016

1 Introduction

This project is our entry to a Kaggle competition, Animal Shelter Outcomes (competition link given in code). In this competition, our primary goal was to become familiar with various data wrangling techniques on date-time, text and categorical data. We did this by applying different ensemble and non-ensemble algorithms (K Nearest Neighbors, Adaboost, Random Forest, Logistic Regression, and Gaussian Naive Bayes). Afterwards, we tuned the parameters of the mentioned algorithms and assessed their predictive power. While we experimented with different algorithms, we also performed simple feature engineering on our data set to look for potential improvement. We concluded the project by trying out a Voting Classifier, which aggregates the results of all aforementioned algorithms to arrive at a final answer.

2 Method and Details

We approached the project in a well-ordered fashion: 1) reformatted the data, 2) selected the appropriate algorithms and 3) optimized the parameters of the algorithms.

2.1 Data Reformatting

The data was initially not formatted properly. As mentioned in the introduction and on the Kaggle competition page, the data comes from the Austin Animal Center. There are 26,729 entries in the training set and 11,456 in the testing set that were recorded from October 1st, 2013 to March, 2016. For example, the training set was comprised of ten features, and the testing set had eight features with a different ordering. Of the dimensions in the two sets, the ('AnimalID') and ('ID'), arbitrary IDs given to each animal, were ruled out for adding unnecessary dimensionality to our model. 'OutcomeSubtype' a feature that only appears in the training set but not the test set, was also removed. Additionally, the target ('OutcomeType') was included in the training set as the outcome, which was moved to a separate set. We then reordered both sets columns to be in the form: ('Name'), ('DateTime'), ('AnimalType'), ('SexuponOutcome'), ('AgeuponOutcome'), ('Breed') and ('Color').

Subsequently, we reformed a number of messy string-type features. 'Name' was transformed into a categorical variable of 0 and 1, with 0 for nameless animals and 1 for named animals. Originally formatted as 'Month-Day-Year Hour:Minute', 'DateTime' was replaced by 4 new numerical

features: 'Year', 'Month', 'Hour' (what hour of the day), and 'WeekDay' (what day of the week). 'SexuponOutcome' contains the sex of the animal and their status together in one string, so we decided to split this feature into two categorical variables: 'Intact' ('Male', 'Female') and 'Sex' ('Neutered', 'Spayed', 'Intact'). Instances with 'Unknown' value got a blank string for the two new variables. 'AgeuponOutcome' represents the age of the animal, either in weeks, months or years. We standardized this variable by converting every animal's age into days (numerical number). 'Breed' tend to include multiple breeds for each animal instance. We only took the first breed if there are more than one, and made an extra indicator variable 'Mix' to specify whether an instance is a mix. The same technique was applied to 'Color', taking only the first color among the potentially different colors an animal may have. Finally, we went through and encoded all of the mentioned categorical features from string type (if needed) to numerical values readable by the algorithms.

At the end, the data was left in the form: ('Name'), ('AnimalType'), ('Age'), ('Breed'), ('Color'), ('Year'), ('Month'), ('Hour'), ('WeekDay'), ('Intact'), ('Sex') and ('Mix').

2.2 Algorithm Selection

For this competition, submissions were evaluated using a multi-class logarithmic loss algorithm, which heavily punishes confident predictions (100% for one label and 0% for others). Therefore, we focused our search for algorithms that return probabilistic predictions – namely, Random Forest, Boosting (AdaBoost), Logistic Regression, Extra Trees and Naive Bayes (in particular, Gaussian NB). Also, we experimented with Bernoulli Naive Bayes and Passive Aggressive Classifier, yet both severely underperformed in our initial tests. Despite being a confident classifier, K Nearest Neighbors was used as a simple baseline to judge other algorithms' effectiveness, because of it's consistent accuracy in our initial tests.

2.3 Algorithm Optimization

In a manner similar to Grid Search, we trained each algorithm on different sets of parameters, collected the average 10-fold cross validation (CV) scores, and selected the set of parameters that yielded the best accuracy with reasonable trade-offs (subject to the specificity of the learning algorithms).

2.3.1 k-Nearest Neighbors

The first algorithm we tried is k Nearest Neighbors. We tested values of `k` ranging from 1 to 26 along with 50 and 100 as potential outlying `k`. We also alternated between `uniform` (all neighbors treated equally) and `distance` (closer neighbors are more heavily weighted) weights for calculating nearest neighbors. Here are the best 5 sets of parameters after training on the training set:

	n	weights	accuracy
1	18	distance	0.6505304381
2	25	distance	0.6503056766
3	50	distance	0.6498571907
4	23	distance	0.6495946548
5	20	distance	0.6494457805

We want to pick a `k` large enough to minimize the noise, and small enough to retain the speed of the algorithm. With the best values of `k` ranging from 18 to 50, we decided on 25 as a reasonable median. We used `distance` as the optimal parameter because of its popularity in the top 5 sets of parameters.

2.3.2 Random Forest

The next popular algorithm used by many Kaggler competitors is Random Forest (RF), due to its minimization of the Log-loss function previously mentioned. In simple terms, RF, a strong learner, is an ensemble of decision trees, weak predictors that perform slightly better than random guessing. At each iteration, a subset of the training data is sampled with replacement, then iteratively split by a random subset of features. When a new data point is presented to model, it runs down all the trees until a terminal node(s) is reached. The resulting label of the new data point is be an average or weighted average of all of the terminal nodes that are reached – or in the case of categorical variables, a voting majority. We proposed a set of different parameters to test Random Forest on:

`n_estimators`: 10, 50, 100, 500, 1000; `max_depth`: 1, 2, 3, 4, 5, 10, 20, 50, None;
`max_features`: 0.1, 0.25, 0.5, 0.75, 'sqrt', 'log2', None; `criterion`: 'gini', 'entropy'.

With 10-fold CV, here are the 5 best-performing sets of parameters for RF:

	n-estimators	max_depth	max_features	criterion	accuracy
1	500	20	0.5	gini	0.6852849038
2	1000	20	log2	gini	0.6851358055
3	1000	20	0.25	gini	0.6849121228
4	500	20	0.25	gini	0.6848376922
5	1000	20	0.5	gini	0.6848354103

Clearly, 20 and gini are the best parameters for `max_depth` and `criterion` respectively. The choice of `n_estimators` between 500 and 1000 are negligible in terms of accuracy (although bigger is better to a certain extent), so we decided to go for 500 to save on training time. When `max_features` = 0.5 (which means 50% of the features are randomly selected for each tree), the model preforms slightly better than 0.25, so 0.5 was picked. In conclusion, the best parameters for RF follow:

`n_estimators` = 500 `max_depth` = 20 `max_features` = 0.5 `criterion` = *gini*

2.3.3 AdaBoost

We also experimented with AdaBoost, a popular boosting algorithm. (We attempted to use XGBoost, but we ran into some trouble installing the package on the department machines). AdaBoost works comparably to Random Forest by training a number of weak learners with each subsequent learner added to the ensemble while trying to correct the misclassifications of the previous learner(s). Parameters that were considered follow.

`learning_rate` : 0.01, 0.1, 0.5, 1.0, 10.0, 50.0, 100.0; `n_estimators` : 10, 50, 100, 500, 1000

Accuracy after searching in the parameter space, with 10-fold CV:

	learning_rate	n_estimators	accuracy
1	0.5	100	0.6512797678
2	0.1	500	0.6492582348
3	0.1	1000	0.6480993291
4	0.5	50	0.6473874807
5	1	50	0.6421130253

There is a trade-off between `learning_rate` and `n-estimators`. Higher `n-estimators` means there are more decision trees in the ensemble, which reduces bias and increases variance. Learning rate shrinks the contribution of each classifier by `learning_rate`, which effectively counters higher `n-estimators`. Accordingly, a large `n-estimates` is effectively tamed by a learning rates from 0.1 to 0.5. The parameter set of 0.5 and 100 performs roughly as well as 0.1 and 500, so since 0.5 and 100 are generally faster to train (fewer estimators), 0.5 and 100 were our final choice of parameters.

2.3.4 Logistic Regression

The last algorithm tested was Logistic Regression (LR), an ensemble algorithm that is comparable to Naive Bayes. Like Naive Bayes, LR is a classifier that takes in a feature x and computes the probability of it being label y; however, unlike Naive Bayes, LR is a discriminative, which means that it models the dependence of an unknown label to a known feature. The parameters tested follow.

`C`: 0.01, 0.1, 0.5, 1.0, 10.0, 50.0, 100.0; `penalty`: l1, l2; `fit_intercept`: True, False; `dual`:True, False; `tol`:0.1, 0.01, 0.001, 0.0001, 0.00001

The best performing parameter sets are listed below.

	C	penalty	fit_intercept	dual	tol	accuracy
1	10	l2	True	True	0.0001	0.643347089
2	10	l1	True	False	0.001	0.6426363613
3	10	l1	True	False	0.0001	0.6425615809
4	10	l2	True	False	0.0001	0.6425615809
5	10	l2	True	False	0.00001	0.6425615809

We decided on `C` $= 10$, `penalty` $= l1$, `fit_intercept` $= True$, `dual` $= False$, `tol` $= 0.001$

2.3.5 Extra Trees

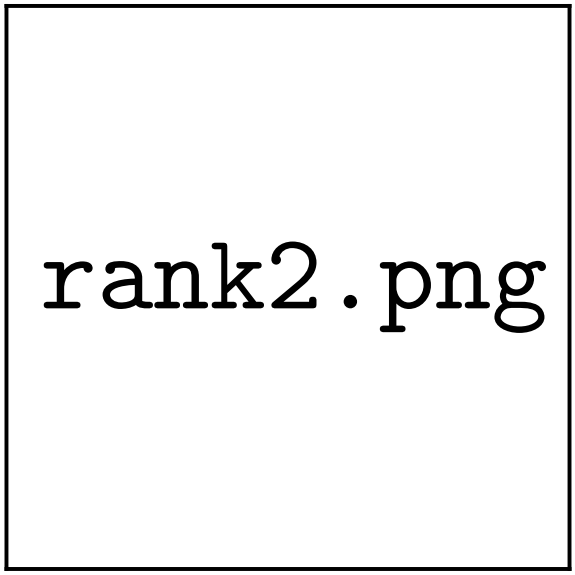
To measure the contribution of individual features to the final prediction, we decided to use Extra Trees Classifier, which is another variation of Random Forest. The key difference in Extra Trees is that, at each variable split, samples are drawn with replacement from the whole population instead of the smaller bootstrap population, and each split is completely random with the possible range of the feature being used. Parameters that we considered:

`n_estimators`: 10, 50, 100, 500, 1000; `max_depth`: 1, 2, 3, 4, 5, 10, 20, 50, None
`max_features`: 0.1, 0.25, 0.5, 0.75, 'sqrt', 'log2', None; `criterion`: 'gini', 'entropy'

The top 5 performing sets of parameters follow:

	n_estimators	max_depth	max_features	criterion	accuracy
1	1000	20	sqrt	gini	0.6773925769
2	1000	20	0.25	gini	0.6769066098
3	500	10	None	gini	0.6767941255
4	1000	10	None	entropy	0.6766824801
5	1000	20	sqrt	entropy	0.676494065

We decided on `n_estimators` = 1000; `max_depth` = 20 `max_features` = sqrt and `criterion` = gini. We then ran the feature importance ranking result from fitting the training data with the target labels, and arrived at the following chart:



3 Results

After collecting the optimal parameters, we decided to try a voting ensemble of all the classifiers through hard voting (taking the majority label) and soft voting (taking the weighted result of individual classifiers). We aimed to compare the training time and accuracy among individual classifiers and the voting ensemble. We present the experiment results in the following table. We concluded that both voting ensembles took as much time as the Random Forest model (discounting time spent to created the input models), but they preformed worst than RF. Random Forest performs best at 0.68, with a pretty long runtime. If speed is what the user strives for, AdaBoost provides a reasonably close prediction, at significantly shorter runtime (15.05 sec compared to 149.94 sec of RF). We hypothesize that XGBoost, if used on this data set, would provide the same predictive power as AdaBoost and RF, at a even faster runtime. Because of the difficulty discussed earlier, we leave this task to the reader to confirm.

	Classifier	accuracy	run time
1	RF	0.684684939	149.9427619
2	ExtraTrees	0.676045243	173.3918262
3	SoftClass	0.6725657256	352.81951499
4	HardClass	0.6709572274	348.491439819
5	AdaB	0.651279768	15.0531311
6	kNN	0.650530438	5.650428057
7	LR	0.642486717	2.668154955
8	GaussianNB	0.580196487	0.263092995

Although the dimensionality of the data set is not particularly large (12 dimensions), we also decided to try reducing the number of dimensions both manually and through Principal Component Analysis (PCA) for potential accuracy gain. According to the Extra tree Classifier, the three least important variables in our data are ('Type'), ('Sex') and ('Mix'). This seems to be counter-intuitive, because these traits of a pet are usually taken into account when its fate is decided. Accordingly, we believed that models susceptible to overfitting overfit to redundant features – such as, (Hour) and (Year). However, with these features removed, the accuracy goes down by 4 percent.

We first tried removing the 3 features above from the training set, kept the remaining 9 the same and ran the learning algorithms again. Then we tried removing the 3 worst performing features by representing the training data in 9 and 7 principal components instead of 12.

	Classifier	accuracy	acc w/o 3 features	acc w PCA = 7	acc w PCA = 9
1	RF	0.684684939	0.6835632905	0.6688986055	0.6774651457
2	ExtraTrees	0.676045243	0.6772797556	0.6644098263	0.6743996003
3	SoftClass	0.6725657256	0.6732387348	0.6565537893	0.6618666923
4	HardClass	0.6709572274	0.6706585262	0.6535981540	0.6577133397
5	AdaB	0.651279768	0.6512797678	0.6261380808	0.6268846232
6	kNN	0.650530438	0.6505304381	0.6465269249	0.6478366798
7	LR	0.642486717	0.6424866886	0.6189931771	0.6234077366
8	GaussianNB	0.580196487	0.5801964871	0.6157016256	0.6194036087

As one can see from the table, the original results and the ones without the 3 least important features are pretty close to each other. Although there is no notable improvement, removing 3 features saves on training time, and could be applied to a much bigger data set in the future, with little compromise on prediction accuracy. However, the results of the training set represented with 7 and 9 principal components show slightly worse results by a small margin. We can see the trend that representing the data with smaller number of principal components worsens the accuracy. Therefore, we concluded that PCA doesn't work well with this data set.

4 Conclusions

We initially approached this competition exploratively. We first explored different data wrangling techniques to convert the data into usable form. With the goal of minimizing the logarithmic loss function, we experimented with different algorithms that range from ensemble methods (RF, Extra Trees, Soft and Hard Voting classifers, AdaBoost) to non-ensemble methods (kNN, LR and GaussianNB).

Our results were fairly surprising. The ensemble methods generally outperformed the non-ensemble methods. However, there was a clear ranking of the algorithms despite any transformation of the data. The hard and soft voting classifiers, for example, performed well, but not well enough to trump one of their inputs, RF. In fact, RF routinely outperformed all other algorithms.

We believe that RFs ability to minimize noise and its robustness against overfitting gave it the advantage necessary to outperform the other methods, thanks to boostrap resampling and the randomization in selecting a subset of features on each split.