Matthew Worley
Springboard Capstone Project 1: Final Report
Prediction of Upsets in the NCAA Men's Basketball Tournament

The Problem

Every year in March 64 men's college basketball teams compete in a single-elimination tournament for the national championship. Because of the win-or-go-home format and the often-shocking results, the tournament is nicknamed "March Madness". Every year some strong teams are eliminated early by lower-ranked "underdogs". These "upsets" drive much of the popularity and excitement of the tournament, which is one of the most heavily wagered sporting events in the world. Every year millions participate in bracket contests where they predict a winner for all 63 games, with points awarded for each correct prediction. Many of these contests involve paid fees and cash prizes awarded to the top-scoring players. Because most bracket participants do not predict upsets accurately, better prediction of upsets provides a clear route to gaining an edge over the competition in these bracket contests. The goal of this project is first to build a robust dataset by combining multiple sources of data on NCAA tournament teams and game outcomes, and then to use machine learning algorithms on this dataset to predict upsets in the NCAA tournament.

The Clients

- Over 70 million individuals who complete brackets annually (~$9 billion wagered in 2016)
- Individuals placing bets on NCAA tournament games
- Expert analysts in sports media who offer tourney guidance via television or web content
- Fee-based NCAA tournament guidance services

Sources of Raw Data

The primary raw data was obtained through an annual NCAA tournament contest hosted by analytics platform Kaggle (https://www.kaggle.com/) Each .csv file is listed below, with a brief description:

TourneyCompactResults – The teams and final scores of all tournament games from 1985-2016
TourneySeeds - Each team's "seed" for the tournament, a 1-16 ranking indicative of team strength
Teams - Unique numeric identifiers for all Division I basketball teams, paired with a team name
TeamSpellings – A listing of possible alternate team names paired with team numeric identifiers
TeamConferences – The conference for each team, listed by season
TeamCoaches – The name of each team's coach, listed by season
TeamGeog – The latitude and longitude of each team's home campus
TourneyGeog – The latitude and longitude of each game location for all previous tournament games.
RegularSeasonDetailedResults – The scores and game-play statistics for all pre-tournament Division I NCAA basketball games from 2003-2017

These files provided sufficient data to identify upsets and build some team features, but also had limitations with respect to feature coverage. This data did not include player-level data to account for team-wide experience, player size, and star players. The Kaggle data also lacked any "advanced" metrics of team performance that account for team differences in strength of schedule, margin of victory, or tempo. To address these limitations, I acquired supplemental data from the web:

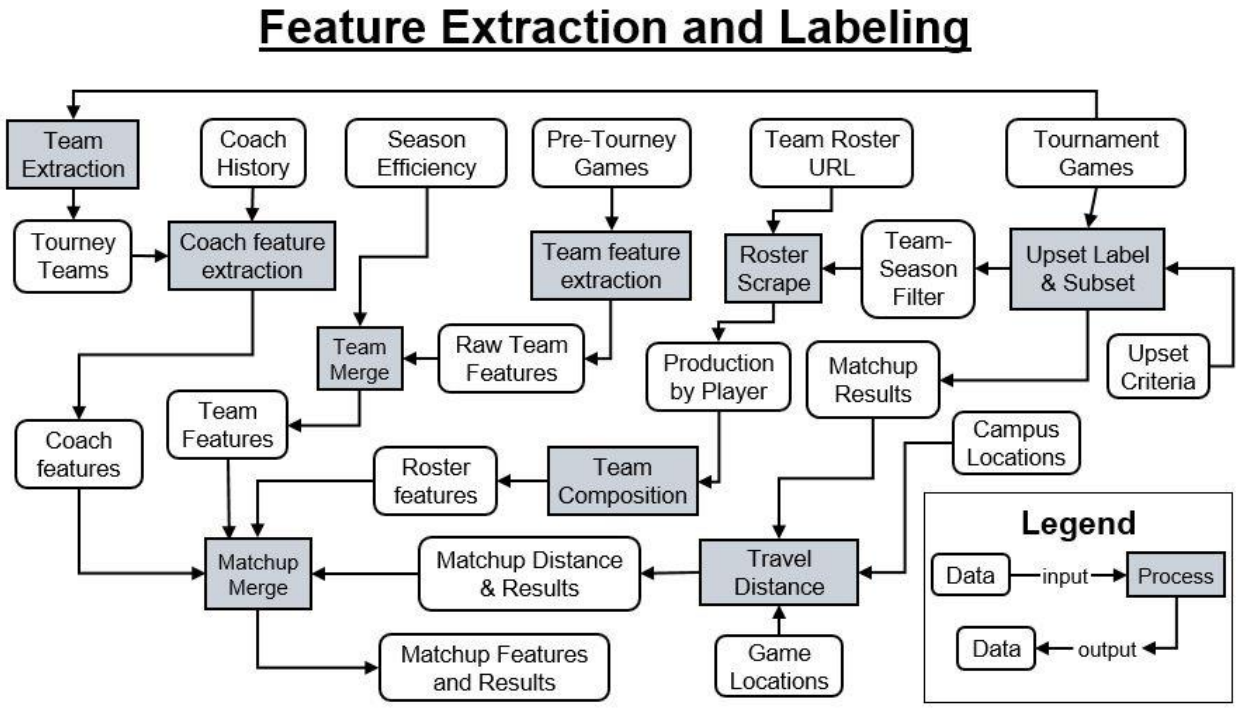
http://kenpom.com/ - Team "efficiency" metrics based on pre-tournament game scores, adjusted for schedule strength and tempo. A separate .csv file was downloaded for each season from 2002-2017.

Using the python *BeautifulSoup* package, I built scrapers to gather over 5,000 individual tables of data from college basketball websites. Each site provided distinct data as described below.

http://www.sports-reference.com/cbb/ - Two player tables, with player-level descriptive data (class, position, height) and performance statistics recorded during basketball game play.
http://www.espn.com/mens-college-basketball/ - Player table with performance statistics
http://www.sports-reference.com/cbb/seasons/ - A table listing each team and season-long statistics, including strength of schedule rating
http://www.basketball-reference.com/awards/all_america.html - Web page listing the 15 Associated Press All-Americans (an award selected by media vote to outstanding players) for each season

Data Wrangling

The goal of data wrangling was to identify upsets in NCAA tournament games and generate features on the teams involved in each game. Each data file used for feature generation required processing in the form of transformation of structure or computation of new variables, before merging with other features for the final feature set used in machine learning. The diagram below depicts the full data wrangling pipeline used to extract features and labels.



Because full details of the data wrangling pipeline are beyond the scope of this report, below I briefly describe a few of the more complex data wrangling problems and solutions I used. Specific examples of data wrangling problems and solutions can be viewed here:
https://github.com/mworles/portfolio

Game selection and upset labels. I treated this project as a classification problem with the goal of classifying future games as upsets or not. To train the model, tournament games needed to be labeled as "upset" or "not upset" according to some criteria set on the teams and the game outcome. To set these

criteria I used the team "seeds", a 1 (best) to 16 (worst) ranking assigned to each team prior to the tournament. Historically, in matchups where the seed difference between teams is at least 4, the team with the lower numerical seed wins at much higher rates than in matchups with a lower seed difference. Model training and testing was restricted to NCAA tournament games with "upset potential", defined as a matchup between teams with a seed difference > 3. For each game, wins by the "underdog" (team with the numerically higher seed) were labeled as 1 to indicate upsets, and all other games were labeled as 0.

"Fuzzy matching" player names. Player-level data obtained from two separate sources (sports-reference and ESPN) required merging prior to generation of features. An inner join on team, season, and player name left 8% of names (nearly 600 players) unmatched due to differences in punctuation or nicknames. Using the python *fuzzywuzzy* string-matching package, I wrote a customized function to match the remaining names. When applied over a dataframe of unmatched players, the function selectively searches and matches names between the two sources by using each player's team and season.

```
# create empty list to fill with match results
plist = []
def match_name_team(row):
    try:
        # pull player name, team name, and season from the dataframe
        p = row['Player']
        t = row['srname']
        s = row['Season']
        # set l as values for key (team and season) in unmatched name dict
        # l is list of unmatched names from key
        l = espn_dict.get((t, s))
        # extractOne function takes player name from unmatched frame
        # returns best-matching name from l and a score value (0-100)
        n, scr = process.extractOne(p, l)
        # combine dataframe player data, matched name, and score into tuple
        t =  (p, t, s, n, scr)
        # add player tuple to the list
        plist.append(t)

# apply the function over all rows of the unmatched player name dataframe
nomatch_sr.apply(match_name_team, axis=1)

# create dataframe from the
df = pd.DataFrame(plist, columns=('Player', 'srname', 'Season',
                                  'Player_espn', 'score'))
```

This approach matched hundreds of names without requiring any case-by-case recoding, and provided superior results than using the function out-of-the-box to search over all unmatched names. Below are examples of players matched by the function.

	Player	srname	Season	Player_espn	score
329	Patrick Swilling	tulsa	2014	Pat Swilling, Jr.	75
514	Patrick Mills	saint-marys-ca	2008	Patty Mills	75
435	B.J. Mullens	ohio-state	2009	Byron Mullens	74
86	J.R. Pinnock	george-washington	2005	Danilo Pinnock	74

Feature computation from player-level data. Player-level data was used to compute team-level features. Most features were computed by transforming the player-level dataset and using *groupby* operations to compute distinct feature values for each unique team and season. Below are several examples:

- I wrote a function that used players' average minutes per game to create a "starter" group variable denoting each player as a "starter" or "bench" player. Groupby operations with the "starter" variable computed points scored separately for a team's starters and bench players.
- I converted player class from string-type data ("SR", "JR", "SO", "FR") to a numeric experience indicator denoting each player's years of college experience. Team-level experience was computed, separately for the whole team and for the starters (the five players who tend to play the most).
- I converted player position from string-type data ("G", "F", "C") to a positional indicator grouping players into "frontcourt" or "backcourt" positions, which allowed computation of position group-specific performance features for each team (points scored, rebounds, assist/turnover ratio).

"Fuzzy matching" team names. Each data source used different team identifiers, which created challenges when combining the features. To allow clean data integration, I created a table linking all of the unique team identifiers from each source. I used the same approach as when matching player names, by using inner joins to match identical team names, and fuzzy string functions to match remaining team names. In one source the team names included both school names and mascots, which I addressed by stripping words from each string. Below are examples of team names matched by this process.

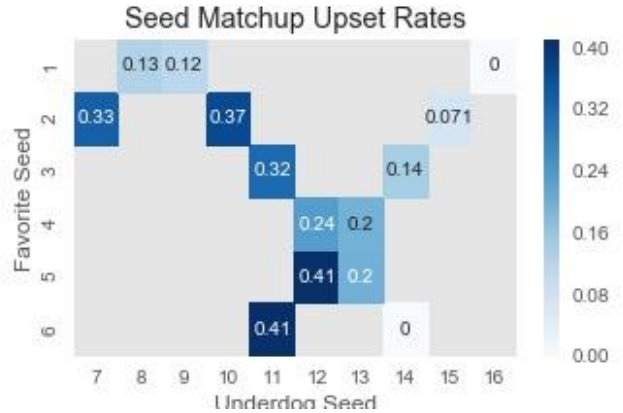
	espn_id	teams	name_spelling	score
310	399	albany great	albany	67
293	315	niagara purple	niagara	67
243	2655	tulane green	tulane	67
174	2473	oakland golden	oakland	67

Data Exploration/Visualization

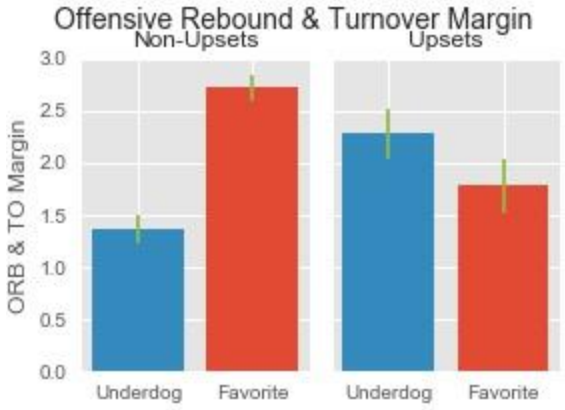
After the data processing stage I completed thorough set of data exploration and visualization exercises. I used data visualization to gain a "high-level" perspective of upsets in the NCAA tournament, by inspecting trends in upsets over time, rates of upsets for individual seeds and seed matchups, and bivariate associations between upset rates and many statistical performance indicators of favorites and underdogs. Although a full description of all the data visualization results cannot be included here, I describe some key findings below, with more detailed results available here:

https://github.com/mworles/portfolio/notebooks/upset_visualization.ipynb

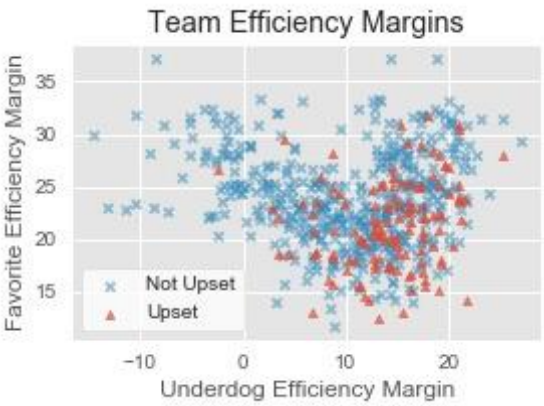
With data visualization I tried to identify some features that could potentially help predict upsets in statistical models. For each game, one team was designated as the "favorite", and one as the "underdog". With data visualization I tried to distinguish favorites who win (not upset) from those who lose (upset), and underdogs who win (upset) from those who lose (not upset). The examples below highlight some team features that were identified as strong candidates for predicting upsets in statistical models.



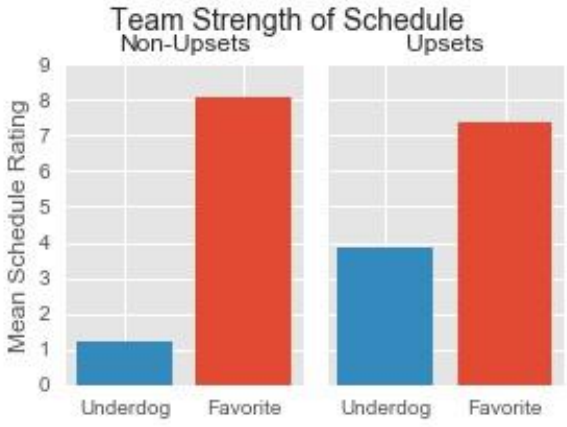
The probability of an upset depends on the seed matchup. Matchups with the highest upset rates include 11 vs 6, 12 vs 5, and 10/7 vs 2.



Upsets involved underdogs with higher pre-tournament margins in offensive rebounds + turnovers, and favorites with lower margins



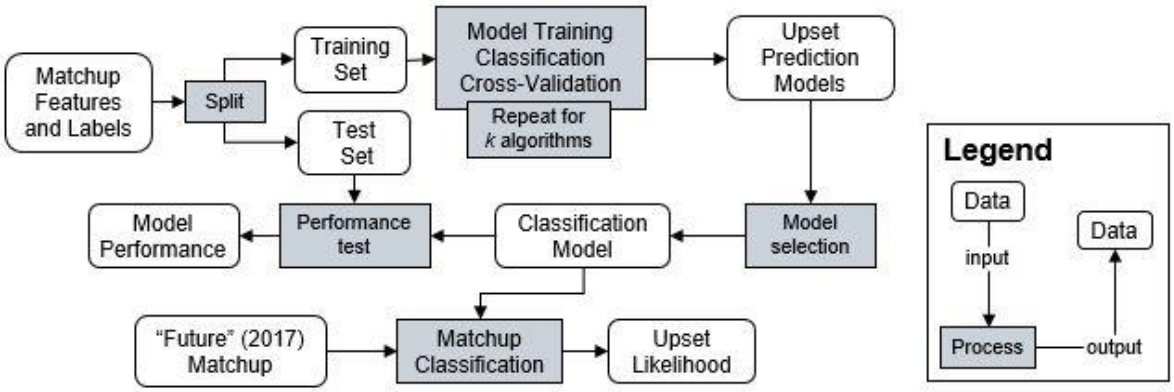
Most upsets involve underdogs with a higher efficiency margin (above 10), and favorites with a lower efficiency margin (below 25)



Upsets involved underdogs who played a more difficult pre-tournament schedule

Machine Learning Classification

The machine learning pipeline for this project is shown below:



First, the data was split for training and testing, with a random 20% of tournament games held out as a test set. I then executed an identical model training process for multiple algorithms. In addition to varying the algorithm, I wanted to test different sets of features used in model training, to determine whether the time invested in gathering supplemental data and generating features did add value in the form of better upset predictions. Four feature sets were used:

Set	Size	Description
"Set 1"	6	Seed, offensive efficiency, and defensive efficiency (both underdogs and favorites)
"Set 2"	11	Set 1 plus total offensive rebound/turnover margin (underdogs & favorites), strength of schedule (underdogs & favorites), and coach success (favorite only)
"Set 3"	41	Set 2 plus 30 additional features identified from data visualization and theory
"Set 4"	164	All available features

Model selection with hyperparameter grid search

For each algorithm and feature set, I conducted a hyperparameter grid search with *k*-fold cross-validation, testing *k*=5 and *k*=10. The grid search identified the best hyperparameters for each algorithm with mean cross-validation *F1* score used as the selection criteria. I used *F1* instead of other scores (accuracy, *ROC AUC*) because the unbalanced labels (78% non-upsets) skewed classification heavily towards true negative predictions. The *F1* score, a balance of precision and recall, is also a useful metric for the real-world scenario of bracket contests, where the "default" choice is a non-upset: one would be 78% accurate choosing non-upset every time. A useful upset prediction model therefore needs to optimally balance recall (predicting as many true upsets as possible) and precision (limiting false positive predictions). The hyperparameter grid search sought the hyperparameters that produced the highest F1 score for each unique combination of algorithm, feature set, and *k*, with higher *F1* scores indicating greater combined average (i.e., harmonic mean) of precision and recall.

Two algorithms were tested: regularized logistic regression and random forests, and both were tested for the four feature sets shown above. The full hyperparameter grids used in the search are shown below, followed by grid search results, shown only for *k* = 5 (*k* size had little impact on scoring).

Regularized logistic regression

Hyperparameter search grid: {'penalty': ('l1', 'l2'), 'C': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]}

Feature set	Cross-Validation F1 Score	Penalty	C
49	0.516	l2	0.001
164	0.459	l2	0.0001
11	0.457	l2	0.0001
6	0.171	l2	0.0001

Random forests

Hyperparameter search grid: {'bootstrap': [True, False], 'min_samples_leaf': [1, 10, 20], 'n_estimators': [5, 10, 50], 'min_samples_split': [10, 20, 40], 'criterion': ['gini', 'entropy'], 'max_features': [2, 6, 11, 24, 49, 164], 'max_depth': [10, 20]}

Feature set	Validation F1 Score	Criterion	Bootstrap	Max Number Trees	Max Tree Depth	Max Features for Split	Minimum Samples for Split	Minimum Samples Leaf Node
11	0.35	entropy	True	5	20	6	10	1
49	0.345	entropy	False	5	10	49	10	1

164	0.334	entropy	False	50	10	164	10	1
6	0.326	gini	False	5	20	2	10	1

The best logistic regression score (0.513) was far superior to the best random forest score (0.35). The range of logistic regression scores (0.17 – 0.51) were much broader than the random forest scores, which fell in a narrow range (0.326-0.35) regardless of the feature set or model hyperparameter combination.

Test set performance

Using the held-out test set of 20% of tournament games, I compared classification performance for the best 8 models from the hyperparameter grid search. Each model was fit to the training set (475 examples) and used to generate predictions on the test set (109 examples). The training and test sets had similar proportions of upsets, which were unbalanced in both sets (Training Set = 22% of all games were upsets, Test set = 23% were upsets). The training and test classification scores are shown below, sorted from highest to lowest *F1* score, with accuracy, precision, and recall scores also displayed.

Algorithm	**Feature set size**	**Accuracy Train|Test**	**Precision Train|Test**	**Recall Train|Test**	**F1 Train|Test**
LogisticRegression	164	0.65|0.68	0.35|0.39	0.77|0.68	0.49|0.49
LogisticRegression	49	0.73|0.69	0.42|0.35	0.71|0.44	0.53|0.39
RandomForestClassifier	49	0.95|0.72	0.89|0.38	0.86|0.32	0.88|0.35
LogisticRegression	11	0.76|0.74	0.44|0.40	0.43|0.24	0.44|0.30
RandomForestClassifier	6	0.98|0.72	0.98|0.31	0.93|0.20	0.95|0.24
RandomForestClassifier	11	0.93|0.71	0.95|0.27	0.72|0.16	0.82|0.20
RandomForestClassifier	164	0.96|0.68	0.99|0.19	0.83|0.12	0.90|0.15
LogisticRegression	6	0.78|0.73	0.47|0.17	0.17|0.04	0.25|0.06

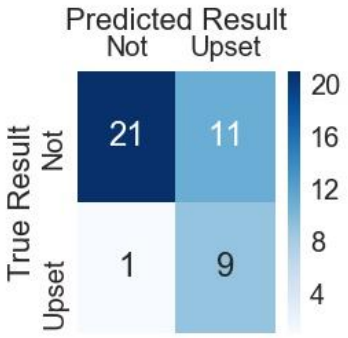
The top model (*L2* Regularized Logistic Regression with $C = 0.0001$ on all 164 features) had similar train/test *F1* scores, while the other 7 models all had significant decline from training to testing. This pattern suggests these models were overfit to the training data, especially the random forest models, which all had training *F1* near 1 but test *F1* near 0.

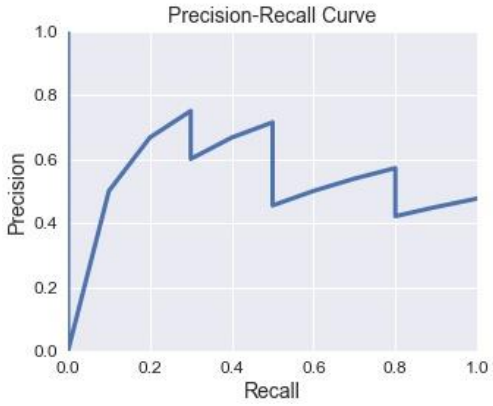
Test set performance for the best model (*L2* Logistic Regression, $C = 0.0001$) suggests this model would be practically useful for predicting upsets. This model had test set recall of 68%. For comparison, in the 2016 tournament the average national prediction rate for underdog winners was only 19%.

Future Matchup Classification

To illustrate prediction of future matchups in a single tournament, I used the model to classify games from the 2017 tournament that took place during implementation of this project. These data were unseen during model training and testing. Classification scores, a confusion matrix, and precision-recall curve are shown below:

Accuracy	ROC AUC	Precision	Recall	F1
0.714	0.778	0.45	0.90	0.60





For the 2017 tournament, the model has very high recall at the expense of precision. The model classifies all true upsets as upsets, but only 10/25 games classified as upsets were true upsets.

Comparison to 2017 Kaggle Contest Entries

To evaluate model performance against an industry benchmark, I used team entries from the 2017 Kaggle March Machine Mania contest, which were posted after conclusion of the 2017 tournament. For this comparison, it is important to acknowledge that the Kaggle contest rules and scoring differ from my project. The Kaggle predictions are probabilities, entries are scored with a log-loss function, and all tournament games are scored. In this project, only games meeting my criteria of "upset-potential" are used in model training and in classification. For the specific task of classifying upsets, these differences may bias performance of my model against the Kaggle competitors. Still, the Kaggle entries provide a reasonable competitive benchmark from other data scientists using machine learning to predict NCAA tournament games.

The full Kaggle contest entry file contains 1,747,226 rows, with each entry containing a prediction for every potential matchup in the tournament. Each entry includes 3 columns where column 1 is an entry identifier. Column 2 is a game matchup identifier in the format "Year_Team1_Team2" where Team1 is the team with the lower numerical identifier. Column 3 is prediction in the form of a probability that Team 1 wins the game. Below is an example of one row from a Kaggle entry:

	user_key	Id	Pred
1534	1111111	2017_1112_1462	0.695046

For this comparison, my goal was to obtain the distribution of classification scores for all the Kaggle entries, and assess the score of my model in comparison to this distribution. To do this, I needed to select the subset of Kaggle predictions corresponding only to games that my model classified, transform the probabilities to upset labels, and compute distinct classification scores for each entry.

I first created a list of the 42 “upset potential” games in 2017 classified by my model matching the Kaggle game identifier format:

```
['2017_1323_1343', '2017_1137_1452', '2017_1423_1438', '2017_1139_1457', …'2017_1257_1276']
```

Using this list I selected the subset of Kaggle entry predictions matching this list. This step reduced the Kaggle entry prediction table to 32,214 rows (42 games x 767 entries).

To generate upset prediction labels from the Kaggle entry probabilities, I first created a “t1_win” column to classify each row as a win (1) or loss (0) for team 1, using probability > 0.50 as a threshold:

	user_key	Id	Pred	t1_win
1534	1111111	2017_1112_1462	0.695046	1
16111	1111111	2017_1211_1462	0.476024	0

I then merged in a dataframe containing the seed values for each team in the matchup, which were then used to create upset prediction labels (“upset_pred”) for all rows. A shown below positive upset prediction labels are matchups where t1_win = 1 and the t1_seed > t2_seed or t1_win = 0 and t1_seed < t2_seed. Matchups not meeting these criteria are negative upset predictions.

	user_key	Id	Pred	t1_win	t1_seed	t2_seed	upset_pred
1534	1111111	2017_1112_1462	0.695046	1	2	11	0
30684	1111111	2017_1423_1438	0.730363	1	12	5	1
2305	1111111	2017_1116_1314	0.155690	0	8	1	0
16111	1111111	2017_1211_1462	0.476024	0	1	11	1

I then merged in the true upset labels (“upset”) for each matchup. This step provides three vectors I need to compute classification scores for each entry: the unique entry identifier, the predicted upset label, and the true upset label:

	user_key	upset_pred	upset
1534	1111111	0	1
30684	1111111	1	0
2305	1111111	0	0
16111	1111111	1	0

I also added a fourth vector that included the predicted probability that the underdog would win the game, which I needed to compute ROC AUC.

For each of the 767 entries, I computed 5 classification scores (Accuracy, ROC_AUC, Prediction, Recall, F1) from the predicted labels/probabilities and true labels. Individual entry scores were appended to lists to create five lists (one for each score type) of 767 scores (one score for each entry). With these

data in hand, I compared the 2017 classification scores from my model against the distribution of classification scores from the 2017 Kaggle contest:

	Accuracy	ROC AUC	Precision	Recall	F1
L2 Logistic Regression (C=0.0001)	0.714	0.853	0.45	0.90	0.60
Kaggle entries					
Percentile of my model	18.77	92.05	62.71	95.70	99.87
Kaggle contest mean	0.72	0.734	0.352	0.18	0.18
Kaggle contest max	0.857	0.919	1	1	0.667
Kaggle contest 25th percentile	0.738	0.694	0	0	0
Kaggle contest 75th percentile	0.762	0.825	0.5	0.2	0.308

For F1 and Recall scores, my model performance was at the 99th and 96th percentile of the Kaggle entries. For ROC AUC, my model was at the 92nd percentile. For precision, my model was above average. For accuracy, my model was worse than average. Recall that I had used *F1* as the scoring criteria for the hyperparameter grid search in the model selection phase. This decision likely weighted performance of my final model more towards F1 than accuracy, because the influence of true negatives was not much of a factor in the model selection phase. Future projects could test whether the use of different scoring criteria in model selection could improve accuracy & precision.

Return-on-Investment (ROI) Analysis

Could this model be used to achieve positive return on investment? To answer this question, I simulated an ROI scenario for the 2017 NCAA Tournament. For this simulation, I calculated the actual returns under the condition that a $100 "money-line" wager was placed on the model-predicted winner for each upset-potential game. The "money-line" wager is a binary bet on the winner of the game, where a correct gamble on an underdog receives greater return than a correct gamble on a favored team. The rate of return is determined by odds differentials set by the sportsbook. For example, for this year's upset of 2-seed Duke by 7-seed South Carolina, a $100 bet on South Carolina at the +281 odds returned $281, while a $100 bet on Duke at the -350 odds (had they won) would have returned only $29.

To conduct an accurate ROI analysis, I located a web archive of money-line odds and scraped the odds for each NCAA tournament game in 2017. Each correct model prediction was used to calculate the return on a $100 wager, while each incorrect model prediction returned a $100 loss. With returns of $1900 and losses of $1200, the total investment produced a net profit of $700, which was a 17% ROI ($700/$4200 = .17).

Clearly, the $1200 losses from 12 incorrect predictions were offset by the 30 correct predictions. Of note, the returns were imbalanced between the true upset predictions and true non-upset predictions. The 9 "true positive" upset predictions returned $1621, while the 21 "true negative" non-upset predictions returned a total of $279. For the 2017 tournament, the model's high recall at the expense of precision was a major factor in achieving a positive ROI.

Client Recommendations

For bracket contest participants, I would recommend picking Round 1 underdogs in games predicted as upsets by the model. In 2017 these recommendations would have produced a 19/24 = 79% first-round accuracy rate for upset-potential games. More important than the overall accuracy rate, participants would have predicted every first-round upset, and picked up extra points in formats that award bonuses

for underdog predictions. I would also recommend avoid picking a favorite team with any upset prediction to reach the Final Four. In 2017, every favorite involved in an incorrect upset prediction was eventually defeated by another team.

For individual game wagers, I would recommend clients to place bets on upset-potential games according to the model predictions, but to weight the investment budget towards high-confidence upset and non-upset predictions. My model probabilities generally corresponded to money-line odds for the 2017 tournament. The underdogs with lower upset probabilities had larger returns but also accounted for most of the false positive upset predictions. Placing larger wagers on the smaller-return, higher-confidence underdogs/favorites while still spreading smaller wagers across the larger-return, lower-confidence upsets should help clients achieve maximal ROI.

Summary

The goal of this project was to develop a useful model for predicting upsets in the NCAA tournament, which could be potentially be used to offer better prediction guidance to millions of bracket contest participants. In addition to clean data from an annual data science competition, I obtained additional data from the web to create more features for machine learning. I then evaluated two classification algorithms on different feature sets to find the best-performing model, and used the best model to classify out-of-sample data from previous tournaments and from a single tournament in 2017. The final classification results outperformed the typical performance of national bracket contest participants. The results also outperformed entries from a popular NCAA tournament machine learning competition. Based on this evidence of superior predictions as compared to both lay bracket contest participants and other data scientists, the product would be practically useful when seeking to predict upsets in future NCAA tournaments.

Ideas for Future Work

In this project I only tested two classification algorithms, logistic regression and random forests. There are many other algorithms worth testing on this classification problem, such as support vector machines, gradient boosted trees, and neural networks.

By predicting a dichotomous event of an upset or not, I framed the problem for classification, but in future projects I could try framing it as regression. Instead of a dichotomous event, I could try predicting a numeric outcome such as a team's final score or the difference in team scores. Under this approach I could test a different set of algorithms such regularized linear regression or regression trees. Because the final outcome of a game is often due to a few chance events, this framework could lead to more accurate predictions.

Future work could also improve the results by implementing strategies that directly address the classification "imbalance" in the data. Most classification algorithms have optimal performance with "balanced" labels, i.e. when the proportions of positive and negative cases are similar. With NCAA tournament upsets, the labels are unbalanced (only 22% are upsets). In this project I indirectly addressed this issue by using the F1 score for model selection. In future projects, I could test strategies for handling unbalanced labels that improve performance of traditional classification algorithms, or test algorithms developed specifically for unbalanced classification problems.

In this project I focused on upsets, a special subset of games in the NCAA tournament where one team is clearly favored over another. With this focus in mind, I used criteria to select only certain games for

model training and testing. I used the team tournament seeds to identify when matchups involved a "favorite" and an "underdog", and roughly 2/3 of NCAA tournament games met these criteria. One side effect of this choice was that I did not use the remaining 1/3 of games for model training or testing. Using the full set of NCAA tournament games for model training and testing could potentially lead to more accurate models. Future work could test other criteria for upsets, or simply seek to predict outcomes of NCAA tournament games regardless of their upset-potential status. Some might say that any victory by a higher numerical seed should be called an "upset", or that upsets should be defined by some criteria other than seeds (e.g., Vegas point spreads).

A final idea for future work is to create a product or system that enables consumers to access and use the model predictions when picking tournament games in their brackets or placing bets on tournament games. For example, a web application could enable users to select individual games and view estimates of confidence for upsets, and build their own brackets using these insights.