Ariana Maisonet

12 July 2024

DATA 71200 01

Devaney

Final Project: Classifying G-xG

The 2011 film Moneyball tells the story of manager Billy Beane of the Oakland Athletics baseball team and how he applies economic principles to sign players and bring the team to the 2002 and 2003 playoffs on one of the smaller budgets in Major League Baseball. He identifies a player's ability to get to first base as a quality that is imperative to winning. Although I'm no Billy Beane, I chose to look at shooting statistics from the 2023-2024 English Premier League to see if a player's performance could be classified and predicted using machine learning. I found my dataset on Kaggle.com however the original dataset comes from FBref.com, a website with hundreds of detailed spreadsheets on different football leagues from around the world. More specifically, I aimed to predict whether the difference of goals minus expected goals (G-xG) would be positive or not. This difference quantifies how good a player is at using their opportunities to score. Expected goals is a probability that is calculated using the location and angle of the shooter, the part of the body the ball is shot off, the type of attack, the clarity of the shooter's path, and other characteristics.

My data originally consisted of 28 columns. I dropped columns that had no correlation to goals nor expected goals. These columns were players' age, birth month, nation, and one called "Matches" which was entirely populated with the word "matches". Rows which contained no

player information were removed as well. A player's position and Squad were one-hot encoded, adding around 25 columns. A binary column was created by taking values in the column 'G-xG' and assigning a 1 if the value was positive, and a 0 elsewhere. This column was named 'Exceeded xG'. Any columns that could be used to calculate goals or expected goals were dropped. And lastly, the column including the players' names were dropped. All NaN values were replaced by zeros and smoothing was applied to these columns so they could be logarithmically transformed.

The only transformations I performed on my data were logarithmic transformations. After visualizing my features, I noticed the distributions were either already bell-curved and Gaussian, or they were right-skewed. Squaring, cubing, and exponentially raising these right-skewed distributions resulted in a uniform distribution, visualized by single bar histograms. The four features I transformed were total shots, shots on target, shot accuracy percentage, and shots per 90 minutes. Cleaning and transformations were done to the same columns in my training and testing sets.

I experimented using a Gradient Boosted Random Forests decision tree for my first supervised learning model. Random Forest uses the 'soft vote' of several different decision trees to calculate the most probable class. These decision trees are distinct from one another in that they randomly select a subset of features to create best fitting nodes with. Gradient Boosted Random Forest uses the same boot-strapped samples that Random Forest uses to train their decision trees on, however trees are built sequentially so a prior tree's errors can attempt to be revised by the current tree's splitting. The parameters I adjusted in my Gradient Boosted Random Forest decision tree were the max depth, learning rate, and the number of estimators. Using GridSearch, the best max depth was 3, n_estimators of 50, and a learning rate of 0.01. Although

this resulted in a greater accuracy, it resulted in overfitting of the training data. This is a common problem with Random Forest and shows that the default parameters for my Gradient Boosted Random Forest decision tree performed better on my test set. The cross-validation scores for the model without parameter adjustment was a recall of 0.95, a precision of 0.91, and an f-1 score of 0.93, whereas the adjusted model had a recall, precision, and f-1 score of 0.85, 0.86, and 0.85 respectively. The second model experimented with was the k-Nearest Neighbors classifier. Data in the test set is classified by finding the nearest classified data point from the training set. The class that the neighbor belongs to is assigned to the unclassified point. Parameters can be tuned to increase the number of neighbors to be greater than one, where the majority class is assigned to the point through voting. This model did not perform nearly as well as the Gradient Boosted Random Forest decision tree. Prior to parameter tuning, the model scored a recall score of 0.36, a precision score of 0.41, and an f-1 score of 0.38. After GridSearch optimization, the model scored a recall score of 0.38, a precision score of 0.44, and an f-1 score of 0.41. Although there was an improvement in the scores, the model performs worse than assigning one class to every point in the test set. This can be explained by some of the weaknesses of KNN models and the nature of classifying G-xG. The features Player Position and Squad were one-hot encoded, adding approximately 25 binary features. These features are 0 most of the time and made up the majority of the features. The sparse data is the main reason for the model's low cross validation scores.

Due to a greater cross validation score from my Gradient Boosted Random Forest classifier, I experimented using PCA for feature selection with this model. Similar pre-processing steps were performed on the data, except missing values were not replaced on the test set. After calculating the cumulative sum variance, it was found that only one component was needed to

explain 97.55% of variance. Using the same parameters as my best-performing supervised Gradient Boosted classifier, this resulted in a lower accuracy score on the test set. Without PCA, the model scored an accuracy of 0.93 on the test set, and with the PCA method, where n_components was set to two, the model scored an accuracy of 0.71. Both models showed overfitting on the training set, however it's interesting that the use of PCA worsened the accuracy. The number of components was tested even though only one component was needed to explain 95% of the variance because one component led to an accuracy of -0.03. As the number of components increased, the accuracy showed no large improvements. The greatest accuracy achieved was at 12 components, yielding an accuracy of 0.73. The overall decrease in accuracy can be explained by some of PCA's weaknesses. When the dimensionality of the data is reduced, information can be lost or oversimplified.

The three clustering algorithms used to preprocess the data were k-Means, DBSCAN, and Agglomerative clustering. K-Means clustering works by initializing a specified number of cluster centers. Points are assigned to their nearest cluster centers, and the cluster center is reinitialized to the mean of each cluster. These steps are repeated until cluster assignment and cluster centers remain unchanged. Agglomerative clustering works by initializing each point as its own cluster and merge two clusters at a time until there remains a pre-specified number of clusters. Clusters are merged using ward, average, or complete linkage measurements. Lastly. DBSCAN clustering begins by identifying densely populated areas within the feature space and choosing a point. A point belongs to a cluster based on how many points fall within a certain distance of it. If points fail to meet their density quota, they are classified as noise. DBSCAN continues this process with a point that has not been classified until all points belong to a cluster or are considered noise. On non-PCA transformed data, k-Means, Agglomerative Clustering, and DBSCAN had an

ARI of 1, meaning all their clustering matched exactly. For PCA transformed data, the three clustering algorithms also had an ARI of 1. For both PCA transformed and non-PCA transformed data, the clustering algorithms had a silhouette score of 0.77, meaning the clustering results were reasonably good.

Across the three projects, I learned how to prepare my data and use visualizations and scores to interpret the efficacy of my models, parameters, and preprocessing steps. In the first project, I learned how to transform my data based on histograms I created of each feature. This was helpful as I learned that certain feature which shared distributional shapes would benefit from being transformed logarithmically. This resulted in a more Gaussian distribution. The second project taught me about the specific parameters in each model, and how they affect the overall performance. Visualizing feature importance also taught me which features played a role in classifying G-xG. It made sense that Manchester City was the only squad to have any feature importance as they had won this season. If I could do it over, I would have done more pruning to my Gradient Boosted Random Forest classifier to fix the overfitting (at the time I didn't realize that was what was happening). I also would not have chosen the k-Nearest Neighbors classifier due to my features being one-hot-encoded. Instead, I would have used a Naïve Bayes classifier. Hopefully with pruning, I could have improved my PCA accuracy. In my third project, I learned how to experiment with different n_component values. I also learned why an n_component of 1 is a good indicator that one's data should be PCA transformed. It also became clearer to me the limitations of unsupervised learning as well as supervised learning.