Jonathan Cao

L30045431

CPSC 69700 Master’s Thesis

**Thesis Literature Review**

1. Introduction

My project proposal is to create a fantasy football team generator that will directly compete in a live draft against human participants. The objective is to generate a team that will perform well in a fantasy league during the upcoming NFL season. As the draft progresses, the generator will readjust itself to output the next “most valuable” player, depending on those remaining in the pool.

1. Implementation

The application will be written in python and use the common statistical, data modeling libraries for cleaning, modeling, and implementation: Pandas, Numpy, SKLearn, and Seaborn.

This project will be treated as either a regression or classification problem, I have not completely decided on my target/output feature yet. Because there are multiple statistical traits tied to a player, I will be using 2 feature selection techniques to pick out a set of significant features which have the most impact towards one category or another: univariate selection, and feature importance. I will be using 2 different techniques to compare which features seem to be the most important from the two.   
 Because this is a categorization problem, there were 4 possible machine learning algorithms that I was considering to train my model on: Logistic Regression, KNN, Random

Forest, and XGBoost. Data visualization with pairplots was used on my test data selected

features to try to determine what proper algorithm to use based on the graphs generated.

Most of the points on the graphs overlapped one another, and so Logistic Regression was eliminated from consideration due to the requirement of a linear line being drawn clearly through the data sets. Random Forest is usually the recommended algorithm in this scenario, however, training with this algorithm takes a long time due to the nature of the internal tree structures possibly spanning deep down the tree during the training iterations. KNN would be a good algorithm to choose here because it only calculates euclidian distance between the points, thus, not requiring a tree.

For this project, I will be training on both algorithms to see which would present the most accurate model.

1. Current State and Research

**Univariate Feature Selection**: This selection method uses various ‘univariate’ statistical tests such as chi2 to calculate the best features. Each feature is compared to the target variable to analyze the variance and statistical significance between them. This method is called ‘univariate’ because each feature is independently considered from the other ones, meaning combinations of features towards a target variable will not be considered. Each feature is assigned a test score at the end. The SKLearn library in python has an implementation of this selection method that we can easily incorporate.

**Feature Importance Selection:** This selection method will assign an “importance” score to each feature using the feature\_importance property of the model; the higher the score, the more relevant it is towards the target variable. The benefit of feature selection using this model is that it considers all combinations of features, not just “univariate”, so interactions within features can be captured. This selection model can be utilized from either the transformer SelectFromModel or Extra Tree Classifier, both from the SKLearn library.

**Random Forest:** Ensemble learning method that operates by generating a multitude of decision trees at training time. It uses techniques of bagging and feature randomness when building each tree to create an uncorrelated forest of trees each with its own degree of “prediction”. There are 2 versions of the algorithm, one for classification and one for regression. For classification tasks, the output is the class/target that is most selected by all of the generated trees. For regression tasks, the average prediction of each individual tree is returned. Random forests take long to train, as there is the possibility that training iterations may span deep down any given tree. SKLearn offers a RandomForestClassifier tool for us to use when utilizing this algorithm during development.

**K Nearest Neighbors:** Simple, supervised machine learning algorithm that can be used for classification and regression type problems. KNN works by finding the euclidean distances between a target point and all other examples in the data. It then selects the specified number, K, of examples closest to the target and then grabs the most frequent class (or average labels) between the points. The correct K value to choose for the data is selected through numerous trials of the algorithm. KNN suffers as the data size grows. SKLearn does offer an implementation for KNN called KNeighborsClassifier and KNeighborsRegressor.

1. References

Harrison, O. (2019, July 14). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Medium.https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

<https://en.wikipedia.org/wiki/Random_forest>

<https://towardsdatascience.com/feature-selection-using-python-for-classification-problem-b5f00a1c7028>

<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

https://scikit-learn.org/stable/modules/