Unstructured Data Collection

For our sentiment analysis, we used a Reddit API to take a look at user posts over the previous 90 posted in r/NFL (The main reddit hub for speaking about the NFL). At the time these 90 days encompassed the entirety of the NFL season. In order to collect this data, we decided to research other methods to collect data, as the PRAWN crawler for reddit was not working for our intended purpose, as it would not go back more than about 3 days. After research we found that Reddits “pullpush” API worked better to get data more than 3 days back. After data collection, we connected to the MongoDB Server which the data has been previously uploaded to and specified the wanted columns – ”title”,”created”,”selftext”. After being imported there was an adaptation needed specifically for the date which was our response variable to be measured as it was in UTC time and needed to be converted to DateTime. After its conversion to ensure the correct range of time (NFL on-season) looking at the min, max which were weeks 35 to 48 of the current year. With the goal taking sentiment from each individual team we created a dictionary key with the team name and the definition as a list of nouns such as the coach’s name, players, nicknames, city names, and other relevant terms to identify the subject team of the post. The dictionaries were then put into a function which looped every instance of a team mention and saved the team specific records into a dataframe specific to each team. Next Step was to complete the sentiment analysis, where the NLTK Vader Sentiment Analyzer was used to loop through each team's dataframe and get sentiment scores for each observation within those dataframes. We were only interested in the compound scores for each row, which were then averaged together by week to get a compound sentiment score for each team and week. After each team had aggregated values for each week, they were added to a dataframe that included the aggregated sentiment score for each week and team to get a full list for all NFL teams. This dataframe was later attached to our structured data in Excel to form a complete dataset for modeling.

Structured Data Collection

For our structured data we had to search the web for data sets that included more than one week of data. We needed data from all weeks prior to the weeks that we were predicting. We found what we needed from pro football reference, it included team statistics for offense and defensive, along with special teams and point differential. We manually exported each team to excel and cleaned the data for each team individually. Removing duplicate columns, and anything that was summable was also dropped; for example we removed expected points. After this we then added 3 week moving average columns to help account for injuries and any other outside factors that could affect team performance. After creating 3 week moving averages for all summable columns we calculated the cumulative sum of the teams statistics. Both the three week moving

averages and cumulative sum will help predict the point spread of teams in the following weeks.

After cleaning the data we had combined all 32 teams schedules and statistics into one large excel file. We then added team win loss columns using binary variables and proceeded to add a before game played record to help with predicting future games. If you included the current record instead of the record going into the game it wouldn't be a significant predictor because you already had the outcome of the game.

After ensuring that the data is all organized and in the corrected columns we had to ensure the spelling from the unstructured and structured teams matched then uploaded the data set to colab.

EDA

Firstly, we needed to clean and format the quantitative data. Since our data was manually collected, we did not have to deal with large amounts of missing data. Blank values in columns such as TO's were filled with 0's, as a blank would indicate a 0 with how we formatted the data. We had to do some type conversions of columns based on how the data was imported, but there were no changes to the data at this point. We also mapped our game result column to 1 = win and 0 = loss for the purpose of modeling.

With the quantitative data cleaned, we could now focus on joining it with the sentiment data. We merged with respect to team and week to get a sentiment score for each team and each week, and repeated the process for opponents. Next, we created dummy variables for all 32 NFL teams as well as opposing teams in each game. After ensuring that variable types were correct, there were no missing values, and the join worked properly, we exported the data to a csv for modeling.

Modeling

Our first step with the newly joined dataset was to drop the Date column, which is no longer necessary. We then split our data into test and training data by using weeks 2 through 10 for training and weeks 11, 12, and 13 for testing. This resulted in roughly a 75/25 train/test split. We decided to standardize the training and test data using the StandardScaler from sklearn. We only standardized quantitative columns and made sure to leave the dummy variables alone. After ensuring all of the aforementioned processes worked properly and our data was in the desired format, we could move onto modeling.

We decided to fit 4 models: Logistic Regression, SVM, RF, and XGBoost. We used grid search with each of the models. The models, their parameter grids, and the best model output are included below:

Logistic Regression (LR)

```
'C': [0.1, 1, 10],
'penalty': ['l2'],
'solver': ['lbfgs', 'liblinear'],
'max_iter': [1000]
```

Best LR model: ': 1, max_iter: 1000, penalty: l2, solver: lbfgs

Support Vector Machine (SVM)

```
'C': [0.1, 1, 10, 100],
'gamma': ['scale', 0.01, 0.1, 1],
'kernel': ['rbf']
```

Best SVM model: {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}

XGBoost (XGB)

```
'n_estimators': [15, 20, 25, 30],
'max_depth': [3, 5, 7],
'learning_rate': [0.1, 0.01, 0.001],
'subsample': [0.8, 0.9, 1.0],
'colsample_bytree': [0.8, 0.9, 1.0]
```

Best XGB model: n_estimators=25, max_depth=3, learning_rate=0.1, subsample=1.0, colsample_bytree=1.0, random_state = 514

Random Forest (RF)

```
'criterion': ['gini', 'entropy'],
'max_depth': [2, 6, 8, 10],
'max_leaf_nodes'  : [6, 12, 18, 24, 30],
'n_estimators': [15 ,30, 80, 100]
```

Best RF model: criterion='gini', max_depth=10, max_leaf_nodes=18, n_estimators=30, random_state=514

We combined all of these models into an ensemble using soft voting to classify a win or a loss based on probability. We deduced that our ensemble model had a W-L

record of 63-23 using the test data. To analyze specific model metrics, we used the classification report function in sklearn which gives precision, accuracy, recall, and f1 score. We also computed a confusion matrix to visually analyze performance. Next, we had to extract the individual probabilities for each game and append them as a column in our test dataset. We did this so that we could convert each probability into a spread and a moneyline for that particular game. We created a function to convert probabilities into american betting odds, and appended both the team and opponent odds into the test data. We also created a function to convert these same probabilities into point spreads, which were also added to our test set. We used a scaling factor of 25 in this function. A scaling factor is a weight applied to the calculation of the spread that accounts for the sensitivity of change in the spread due to variation in win probability. We also adjusted these odds to include a 2.5% hold, or in other words, 2.5% guaranteed casino profit assuming the public is split 50/50 on a particular bet. Utilizing all of these functions together as well as ensuring the indices matches for each game, we were able to produce a final dataframe that included Week, Game, Team, Opponent, Team and Opponent Win Probabilities, Team and Opponent Moneylines, and Team and Opponent Spreads. We then filtered this dataframe by week and exported each week's data into a csv for external comparison to sportsbook lines.