Machine Learning Analysis

When performing analysis on this dataset, the types of classifiers used must first be determined. Because this was a labeled dataset, supervised machine learning techniques were used. Of the possible supervised machine learning techniques, the random forest classifier, support vector classifier, and neural network were selected. The random forest classifier was selected because of its robustness and ability to perform well on datasets with many features. The support vector machine was selected because it tends to perform well when there is a clear separation between classes. The dataset contains only two classes, games where the blue team won and games where the red team won, where only one team can win, so there is indeed a clear separation between classes. Finally, a neural network was chosen because of its ability to model complex functions. League of Legends games are extremely complex with thousands of factors that can impact the game. Neural networks were chosen because they are likely able to understand how these factors interact and determine the outcome of a game.

Random Forest

The first step in fitting a random forest to the dataset is splitting the dataset into a training and testing data set. This was done using Scikit-Learn's `train_test_split`function. This is shown in Figure 1 below.

```
# First split the data into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(league_wrangled, result,
                                                    test_size=0.33, random_state=42)


print('Training Features Shape:', X_train.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', X_test.shape)
print('Testing Labels Shape:', y_test.shape)
```

```
Training Features Shape: (5079, 5719)
Training Labels Shape: (5079,)
Testing Features Shape: (2503, 5719)
Testing Labels Shape: (2503,)
```

Figure 1. Splitting dataset into training and testing sets.

Next, a RandomForestClassifier class object is created from Scikit-Learn's implementation and then fit onto the dataset.

```
# Import the Random Forest classifier
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=1000, random_state=42)
rf.fit(X_train, y_train)
```

Figure 2. Creating and fitting Random Forest.

The classifier was able to achieve an accuracy of 0.9648. The confusion matrix of the results are plotted in the heatmap in Figure 3.

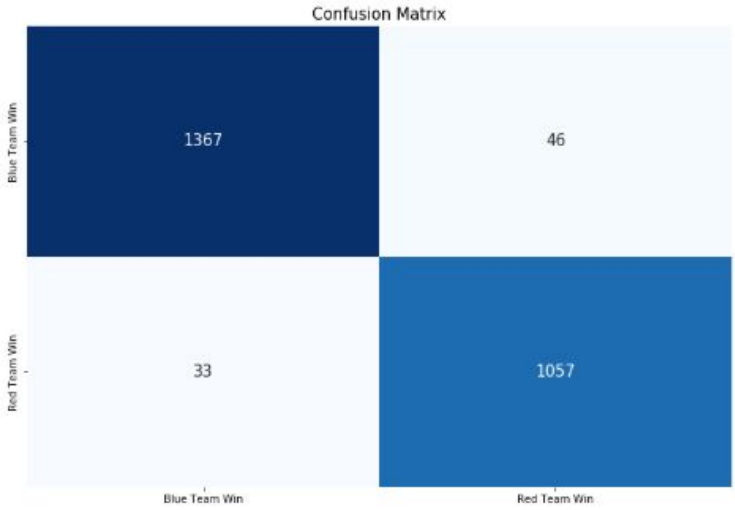

Figure 3. Confusion matrix of results.

From the confusion matrix, the classifier incorrectly predicted only 79 samples out of the 2503 samples within the testing dataset. Additionally, the most significant features were displayed using the random forest classifier's attribute `.feature_importances_`. The five most significant are displayed below in Figure 4.

```
Variable: bInhibs              Importance: 0.0745
Variable: rInhibs              Importance: 0.06558
Variable: rmiddle_inner        Importance: 0.04422
Variable: bbmiddle_base        Importance: 0.04298
Variable: rbmiddle_base        Importance: 0.04043
```

Figure 4. Five most important features in random forest.

Based on the values, it is clear that games are not decided by a single feature alone, but rather a combination of multiple features.

Support Vector Classifier (SVC)

In order to create the support vector classifier, Scikit-Learn's implementation will be used. In order to train the SVC, the training and testing dataset must first be scaled. This is done using Scikit-Learn's StandardScaler() object. The features must be scaled in order to prevent the feature with the largest range from completely dominating the results of the SVC. The SVC is instantiated and trained in the same way as the random forest classifier: by first instantiating the SVC object and calling the fit method on the object. This is shown below in Figure 5.

```
# Import SVM
from sklearn.svm import SVC

svc = SVC(gamma="scale")
svc.fit(X_train_ss, y_train_ss)
```

Figure 5. Creation and Training of Support Vector Classifier.

The SVC was able to achieve an accuracy of 0.9373, performing slightly worse than the random forest.

Neural Network

Due to how complicated neural networks can be, and the number of features, a simple network architecture is created first. This is done using TensorFlow's Keras API. The architecture is shown in Figure 6.

```
Model: "base_league_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 5719)]            0
_________________________________________________________________
dense_302 (Dense)            (None, 2859)              16353480
_________________________________________________________________
dense_303 (Dense)            (None, 1)                 2860
=================================================================
Total params: 16,356,340
Trainable params: 16,356,340
Non-trainable params: 0
_________________________________________________________________
```

Figure 6. Simple Neural Network Architecture.

The neural network is then trained on the scaled training and testing data. This model performed the worst, at an accuracy of 0.56 and loss of 6.677.

Improving models

The three models used in this analysis all contain many hyperparameters capable of affecting the performance of each. To optimize their performances, GridSearchCV is used. This is done by creating parameter grids, which GridSearchCV iterates over to determine which combination of parameters results in the best score. The parameter grids are shown in Figure 7 below.

```
from sklearn.model_selection import GridSearchCV

rf_parameters = {
    'n_estimators':[100,200,500,1000],
    'criterion':['gini', 'entropy'],
    'max_features':['auto', 'sqrt', 'log2', None]
}

svc_parameters = {
    'C':[0.001, 0.01, 0.1, 1, 10],
    'gamma': ['auto', 'scale']
}

nn_parameters = {
    'dropout_rate': [0.5, 0.2],
    'neurons': [1000, 250],
    'batch_size': [32, 8],
    'epochs': [10, 100]
}
```

Figure 7. GridSearchCV parameter grids.

Using GridSearchCV, the Random Forest Classifier's accuracy increased from 0.9684 to 0.9812, while the Support Vector Classifier's accuracy decreased to 0.9368 from 0.9372. Due to time constraints, it is difficult to test neural network architectures, but it appears that it would require a much larger neural network in order to accurately classify the game winner based on the results of GridSearchCV. The neural network, despite having 4 additional layers, was only able to reach an accuracy of 0.524.