머신러닝 알고리즘을 활용한 온라인 게임 승부 예측

2023-13469 김태진

1. 서론

(1) 주제 선정 배경

코로나 19로 인한 사회적 거리두기가 본격적으로 시작되면서 온라인 게임산업은 엄청난 성장을 보였다. 게임 이용자수와 함께 게임 중계 방송 등 파생된 산업에서도 많은 변화가 있었다. 특히 온라인 게임 중계 중 가장 인기가 많은 리그 오브 레전드 챔피언스 코리아(이하 LCK)는 평균 분당 시청자수가 2022년 대비 22%나 증가했다. E-스포츠 역시 기존의 스포츠와 같이 응원하는 팀과 선수들을 보기 위해 많은 사람들이 모이고 있다는 것이다. 관객들이 가장 궁금해하는 것은 최종적으로 경기의 결과이다. 따라서 본 과제에서는 머신러닝 알고리즘을 활용하여 온라인 게임의 승부를 예측해보고자 한다.

(2) 해결하고자 하는 문제

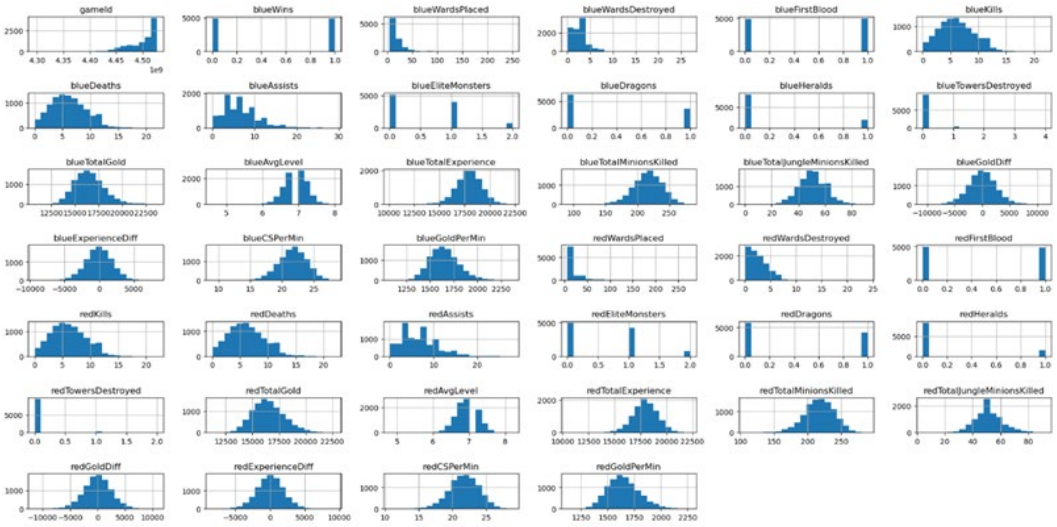
본 과제에서 분석할 온라인 게임은 라이엇 게임즈의 "League of Legend"이다. 국내에서는 2023년 8월 기준 PC방 점유율 39.77%로 1위를 기록한 국내 최고의 인기게임이며, 게임산업을 주도하고 있는 게임 중 하나이다. 본 게임은 레드팀과 블루팀 각각 5명의 플레이어가 각자의 포지션을 맡아 '넥서스'라 불리는 맵 상의 상대의 진영을 무너뜨리면 승리하는 AOS형 게임이다. 게임 맵 안에서 승부에 영향을 줄 수 있는 변수들은 정말 다양하다. 게임 내에서 미니언이나 몬스터를 처치할 시 경험치가 쌓이며, 이 경험치는 플레이어의 스킬과 역량을 강화할 수 있다. 또한 몬스터 처치 시 게임 내에서의 화폐인 골드가 쌓여 이를 통해 아이템을 구매하여 강해질 수 있다. 이외에도 포탑의 체력, 처치한 드래곤과 바론, 상대를 처치한 횟수, 처치된 횟수 등 다양한 요소가 존재한다. 따라서 본 과제에서는 게임 시작 이후 10분이 지난 후의 다양한 변수들의 데이터를 학습하여 최종 게임의 승패를 예측하는 "이진 분류"(블루팀 또는 레드팀) 문제를 해결하고자 한다.

2. 본론

(1) Dataset

본 과제에 사용되는 데이터는 게임 시작 이후 10분 뒤의 게임 상황에 대한 전반적인 데이터를 담고 있다. 데이터의 각각의 게임은 독립적으로 진행되었다. 레드팀과 블루팀 각각에서 19개의 feature를 담고 있고 따라서 총 feature의 수는 38개이다. 또 각

게임의 승패의 정답인 blueWins 열의 1일 때 블루팀의 승리, 0이면 레드팀의 승리를 의미한다. 따라서 본 과제에서는 label을 blueWins로 설정하였고, 나머지 열들을 features로 고려하였다. 또 이해를 돕기 위해 게임 속 데이터의 용어에 관한 설명을 Appendix에 첨부하였다. 데이터의 출처는 Kaggle 웹사이트의 "League of Legends Diamond Ranked Games (10 min)" dataset을 활용했으며, 이는 Riot Games에서 제공한 데이터로 데이터의 품질을 보장할 수 있다.



Dataset의 각각의 features의 분포를 나타낸 사진으로, gameid와 blueWins, 38개의 게임 내의 변수들을 포함해 40개의 변수의 분포는 위와 같다.

(2) 데이터 정규화 작업

각 특성의 크기 범위에 있어서 차이가 생기면, 예측 오차가 커질 가능성이 높다. 또한 최적점에 수렴이 느리고 어려워진다. 따라서 데이터의 sklearn.processing의 Standard Scaler를 사용하여 데이터를 정규화시켰다.

(3) 분류 모델

본 보고서에서는 승률을 예측하기 위한 총 3가지의 분류 모델을 적용했다. 로지스틱 회귀 모델과 랜덤 포레스트 모델, 소프트 벡터 머신 모델을 활용하였으며, 각각의 알고리즘의 이론적 배경은 다음과 같다.

(a) 로지스틱 회귀 모델 (Logistic Regression)

로지스틱 회귀는 로짓 변환에 대한 선형 회귀이다. 로짓 변환의 역함수인 로지스틱 함수를 이용해 이진 분류 문제에서 양성 클래스에 속할 확률을 예측하는 모델이다. 본 데이터에 대해서 양성 클래스(BlueWins = 1)에 포함될 확률이 0.5 이상이면 그 클래스에 속한다고 판단하였다.

$$p(BlueWins = 1 \mid X) = \frac{1}{1+e^{-logit(p)}}$$

로지스틱 함수의 손실 함수는 이진 교차 엔트로피 손실함수로 양성 샘플을 0에 가깝게 고려하거나 음성 샘플을 1에 가까이 예상하면 손실함수가 크게 증가하는 형태이다.

$$L(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(p_i) + (1-y_i)\log(1-p_i)$$

손실 함수의 그래디언트 벡터를 구할 수 있기 때문에 경사하강법을 활용하면 최적의 파라미터 변수들을 구할 수 있다.

(b) 랜덤 포레스트 모델 (Random Forests)

랜덤 포레스트 모델은 배깅된 개별 트리들 사이의 상관관계를 없애 성능을 향상시킨 알고리즘이다. 일반적으로 배깅 앙상블은 모든 특성을 모두 고려하여 분할을 결정하여 결정 트리를 학습한다. 영향력이 큰 특성에 대해서 모든 앙상블의 결정트리가 분할을 위해서 그 특성을 사용하기 때문에 트리 간의 상관성이 커지게 된다. 이 경우 배깅 앙상블의 예측 분산이 크게 줄어들지 못한다는 단점이 존재한다.

$$Var\left[f_{bag}(x)\right] = \frac{1}{B^2}\sum_{b=1}^{B} Var[f_b^*(\mathbf{x})] + \frac{2}{B^2}\sum_{1\le i\le j\le B} Cov\left[f_i^*(\mathbf{x}), f_j^*(x)\right]$$

따라서 랜덤 포레스트 모델은 랜덤 패치 방식으로, 원본 특성 중 p개의 일부 특성만을 랜덤하게 선택하여 개별 결정트리를 학습한다. 개별 트리간의 상관관계가 줄어들기 때문에 예측 분산의 감소 효과가 증폭된다.

(c) 선형 서포트 벡터 머신 모델 (Linear SVM)

분류에 유효한 결정 경계들 중 테스트 데이터가 입력되었을 때 높은 분류성능을 만드는 결정 경계를 구하기 위해 선형 서포트 벡터 머신 모델을 활용한다. Linear SVM은 이진 클래스 사이의 마진을 가장 크게 만드는 결정 경계를 찾는 모델이다. 이때 마진은 결정경계에서 가장 가까운 샘플까지의 거리를 말한다. 마진을 최대화하는 문제는 결국 파라미터 벡터의 L2 norm의 제곱에 대한 최소화 문제로 변형된다. 제약조건에 있는 상태에서 L2 norm을 최소화하는 문제는 라그랑주 승수법을 이용해서 풀 수 있으며, 이를 이용해서 주어진 데이터셋의 최대 마진의 결정경계를 찾을 수 있다.

$$w = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \ , b = \frac{1}{n_{sv}}\sum_{i\ in\ sv} (y_i - w^T \mathbf{x}_i)$$

(4) 분류 성능 및 시각화(혼동 행렬, ROC 곡선,특성 중요도)

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, roc_curve, auc
```

각각의 model은 sklearn에서 LogisticRegression, RandomForestClassifier, SVC를 이용했다. 또 성능 평가를 위해서 classification_report와 accuracy_score를 이용해 정량적으로 모델의 분류 성능을 판단하고자 하였다. accuaracy_score는 test 데이터의 정답과 학습모델의 결과가 같은 데이터의 비율을 의미하며, 0~1사이의 실수값으로 리한다. Classification report은 모델 성능 평가지표를 한눈에 볼 수 있는 표를 제공한다. 각 클래스에서 precision, recall, f1-score, support 그리고 모델 성능에 대해서는 accuracy와 micro avg, weighted avg를 제공한다.

Precision : positive prediction에 대한 정확도 (TP / (TP + FP))

Recall : 예측 성공한 데이터 중 positive 예측의 비율 (TP / (TP + FN))

F1 score : 정밀도와 재현율의 조화평균 (2*recall*precision / (recall + precision))

Accuracy : 전체 중 예측 성공한 데이터의 비율 ((TP + FN) / ALL)

Micro average: 두 라벨에 대한 F1-score의 평균값

Weighted average: 라벨의 개수에 비례하게 F1-score를 가중치평균한 값

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

모델의 분류 성능을 직관적으로 이해하기 위해서 혼동 행렬을 출력하여 데이터 예측 결과를 시각화하였다. 혼동 행렬은 성능 측정을 위한 예측값과 실제값의 분포를 비교한 표이며, 본 과제에서는 이중 분류를 진행했기 때문에 TP,FP,FN,TF의 분포를 나타내었다. 각각의 모델은 sklearn.metrics의 confusion_matrix를 이용하여 혼동 행렬을 얻을 수 있다. 혼동 행렬 시각화 함수는 아래과 같으며 test data와 예측 데이터를 입력하여 혼동 행렬을 얻을 수 있다.

```
# 혼동 행렬 시각화 함수
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(title)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
```

ROC 커브는 이진 분류기의 성능을 표현하는 곡선이다. ROC 커브의 x축은 TPR(=TP/(TP+FN))(True Positive Rate)이고 y축은 FPR(=FP/(FP+TN))(False Positive Rate)로 구성된 곡선이다. 모델의 정확도가 낮을수록 중간선에 가까운 곡선을 나타내며, ROC 곡선의 아랫부분의 넓이를 AUC라고 하며 이 값이 낮을수록 정확도가 낮다고 해석할 수 있다. ROC 커브를 출력하고 AUC 값을 얻는 코드는 아래와 같다.

```
# 랜덤 포레스트 특성 중요도 시각화
def plot_feature_importances(model, features):
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    plt.figure(figsize=(12, 6))
    plt.title("Feature Importances")
    plt.bar(range(features.shape[1]), importances[indices], align="center")
    plt.xticks(range(features.shape[1]), features.columns[indices], rotation=90)
    plt.xlim([-1, features.shape[1]])
    plt.show()
```

또 랜덤 포레스트 모델의 경우 일반적인 배깅 방법의 학습 결과에 대한 예측의 해석력이 떨어진다는 단점이 존재한다. 각 결정트리에서 어떤 특성을 사용한 노드가 지니 불순도를 감소시키는지 알게 된다면, 특성의 중요도를 측정할 수 있다. 또 랜덤 포레스트 모델의 특성 중요도는 각각의 결정트리에서 계산된 특성 중요도를 모두 더한 후 트리 수로 나눈 값이 된다.

$$\Delta G = G - \frac{n_{left}}{n} G_{left} - \frac{n_{right}}{n} G_{right}$$

(a) 로지스틱 회귀 모델

```
# 로지스틱 회귀 모델
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
lr_probs = lr_model.predict_proba(X_test)[:, 1]

print("Logistic Regression Accuracy:", accuracy_score(y_test, lr_predictions))
print(classification_report(y_test, lr_predictions))
```

위 코드를 활용해서 로지스틱 회귀 모델을 학습시킨 결과 결과는 아래와 같이 나타났다.

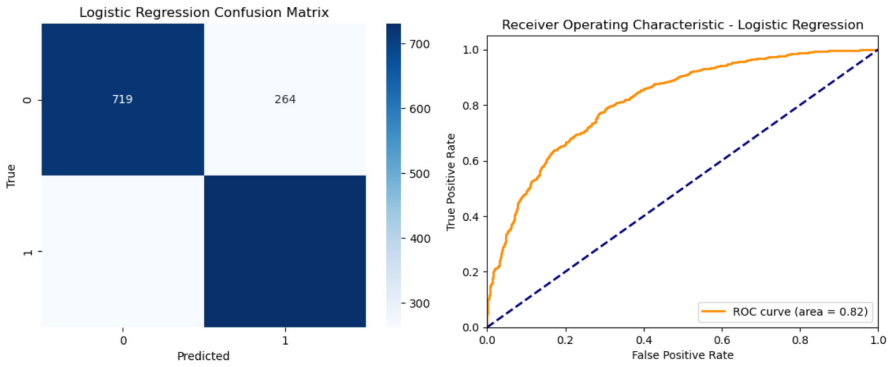
```
Logistic Regression Accuracy: 0.7332995951417004
              precision    recall  f1-score   support

           0       0.73      0.73      0.73       983
           1       0.73      0.74      0.73       993

    accuracy                           0.73      1976
   macro avg       0.73      0.73      0.73      1976
weighted avg       0.73      0.73      0.73      1976
```

로지스틱 회귀 모델은 약 73.33%의 Test data에 대한 예측 정확도를 보였다. 예측 클래스와 실제 클래스를 기반으로 혼동 행렬을 출력해보았을 때 TP(True Positive)와 TN(True Negative)가 두드러지게 나타남을 확인할 수 있다. 로지스틱

회귀 모델의 ROC curve의 아래면적인 AUC 값은 0.82로 나타났다.



(b) 랜덤 포레스트 모델

```
# 랜덤 포레스트 모델
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_probs = rf_model.predict_proba(X_test)[:, 1]

print("Random Forest Accuracy:", accuracy_score(y_test, rf_predictions))
print(classification_report(y_test, rf_predictions))
```

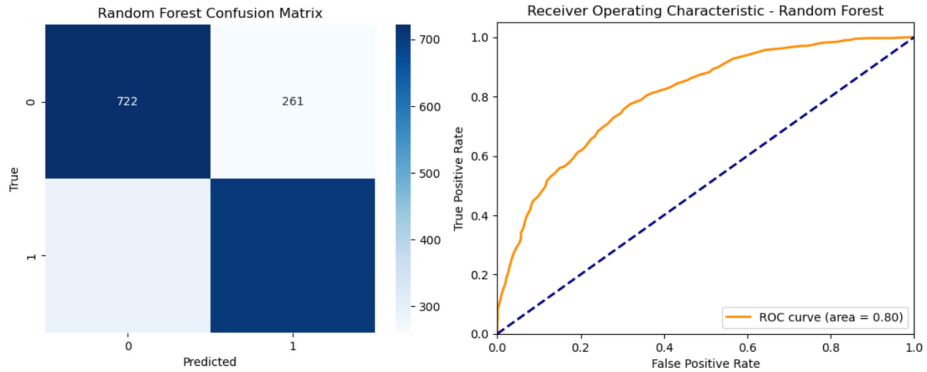
위 코드를 활용해 랜덤 포레스트 모델의 예측 성능과 분류 보고서를 얻었다.

```
Random Forest Accuracy: 0.72165991902834
              precision    recall  f1-score   support

           0       0.71      0.73      0.72       983
           1       0.73      0.71      0.72       993

    accuracy                           0.72      1976
   macro avg       0.72      0.72      0.72      1976
weighted avg       0.72      0.72      0.72      1976
```

랜덤 포레스트 모델의 경우 약 72.17%의 Test data에 대한 예측 정확도를 보였다. 또 예측 클래스와 실제 클래스를 기반으로 혼동 행렬을 출력해보았을 때 역시 TP와TN의 비율이 확연히 높게 나타났다. 랜덤 포레스트 모델에서 ROC curve의 아래면적인 AUC 값은 0.80으로 구해졌다.



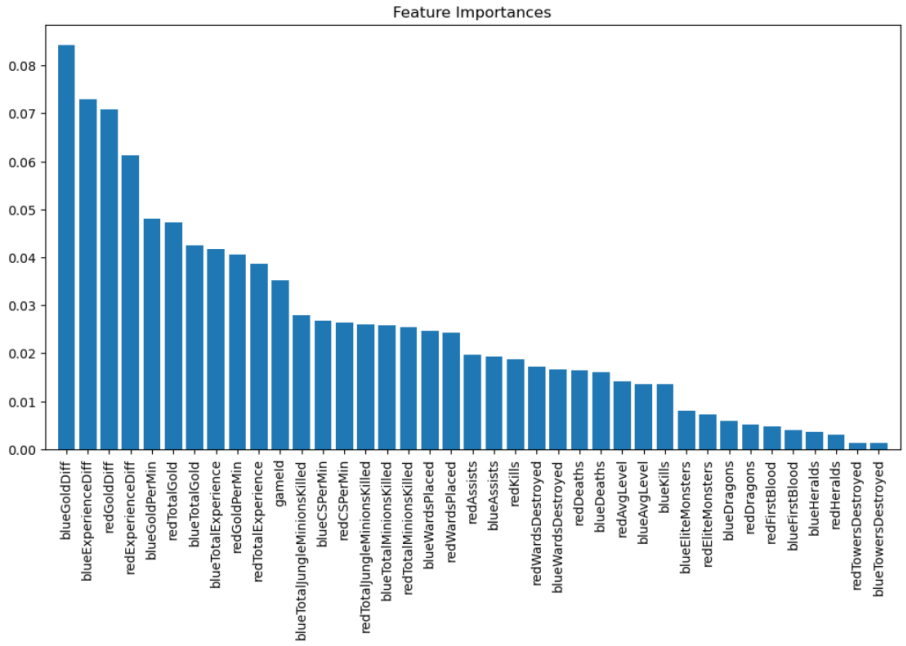
추가적으로 랜덤 포레스트 모델에서는 특성 중요도를 계산하여 38개의 특성 중 어떤 요소가 지배적인 영향을 끼치는지 시각화하여 변수들의 특성을 분석할 수 있었다. feature_importances_ 함수를 활용하여 특성 중요도를 계산하여 중요도 순서대로 정렬하여 어떤 변수가 큰 영향을 미치는지 확인할 수 있었다.

```
# 랜덤 포레스트 특성 중요도 시각화
def plot_feature_importances(model, features):
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    plt.figure(figsize=(12, 6))
    plt.title("Feature Importances")
    plt.bar(range(features.shape[1]), importances[indices], align="center")
    plt.xticks(range(features.shape[1]), features.columns[indices], rotation=90)
    plt.xlim([-1, features.shape[1]])
    plt.show()

# 랜덤 포레스트 특성 중요도
plot_feature_importances(rf_model, features)
```

실행 결과는 아래와 같다.



랜덤 포레스트의 특성 중요도를 분석한 결과 상대팀과의 골드와 경험치 차이가 지배적인 변수로 나타났다. 전체 골드와 경험치와 같이 다른 요소로 인해 변화되는 특성을 합산된 특성을 제외하면 각 팀의 분당 CS(미니언 처치 수)가 중요한 특성임을 확인할 수 있었다. 또 비지배적인 특성으로는 전령 처치 수와 타워 파괴 숫자임을 알 수 있다. 이를 통해 플레이어들은 승리를 위해 게임 시작 후 10분까지 타워를 부수거나 전령 처치에 집중하기보다는 미니언과 상대팀캐릭터를 처치하여 골드와 경험치 차이를 벌려놓는 것이 최종 승률에 더욱 큰 도움을 준다

는 것을 확인할 수 있었다.

(c) 선형 서포트 벡터 머신 모델

```
# 서포트 벡터 머신 모델
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_probs = svm_model.predict_proba(X_test)[:, 1]

print("SVM Accuracy:", accuracy_score(y_test, svm_predictions))
print(classification_report(y_test, svm_predictions))
```

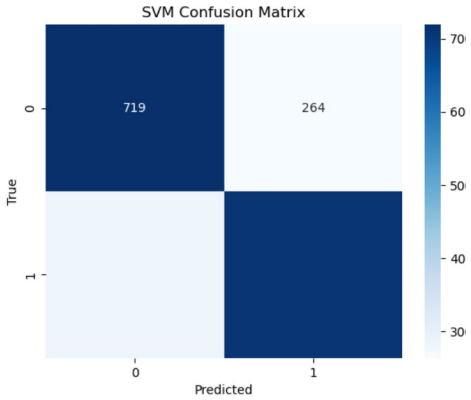
위 코드를 이용하여 선형 서포트 벡터 머신 모델의 예측 성능과 분류 보고서를 출력했다.

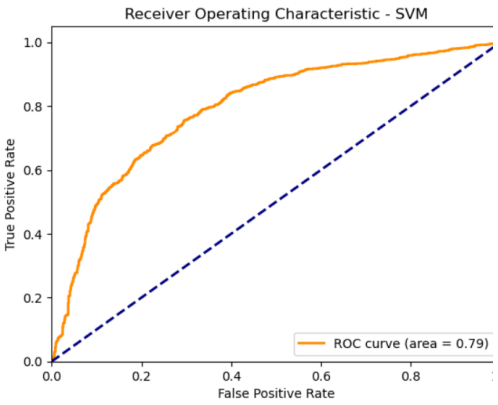
```
SVM Accuracy: 0.7226720647773279
              precision    recall  f1-score   support

           0       0.72      0.73      0.72       983
           1       0.73      0.71      0.72       993

    accuracy                           0.72      1976
   macro avg       0.72      0.72      0.72      1976
weighted avg       0.72      0.72      0.72      1976
```

선형 서포트 벡터 머신 모델을 학습하여 72.27%의 예측 정확도를 얻을 수 있었다. 혼동 행렬을 출력한 결과, TP와 TN이 확연히 높은 비율을 나타냄을 시각적으로 확인할 수 있었다. ROC curve의 아래 면적인 AUC값은 선형 서포트 벡터 머신에서 0.79로 나타났으며 앞선 로지스틱 회귀 모델과 랜덤 포레스트 모델의 정확도에 비해 조금 떨어지는 모습을 보였다.





(5) 앙상블 방법을 통한 예측 성능 향상

앙상블 방법을 통해 단일 모델을 사용할 때보다 예측 분산을 감소시켜 과적합을 방지할 수 있으며, 예측 성능을 더 향상시킬 수 있다. 앞서 진행했던 랜덤 포레스트 모

델에 Gradient 부스팅 모델을 활용하여 예측능력을 향상시켰다.

```
# 그라디언트 부스팅 모델
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
gb_predictions = gb_model.predict(X_test)
gb_probs = gb_model.predict_proba(X_test)[:, 1]

print("Gradient Boosting Accuracy:", accuracy_score(y_test, gb_predictions))
print(classification_report(y_test, gb_predictions))
```

```
# 투표 앙상블 모델 (랜덤 포레스트와 그라디언트 부스팅)
ensemble_model = VotingClassifier(estimators=[
    ('rf', RandomForestClassifier()),
    ('gb', GradientBoostingClassifier())
], voting='soft')

ensemble_model.fit(X_train, y_train)
ensemble_predictions = ensemble_model.predict(X_test)
ensemble_probs = ensemble_model.predict_proba(X_test)[:, 1]

print("Ensemble Model Accuracy:", accuracy_score(y_test, ensemble_predictions))
print(classification_report(y_test, ensemble_predictions))
```

앙상블 방법을 활용한 결과 기존 랜덤 포레스트 모델의 예측성능인 72.17%에 비해 소량 향상되어 Gradient Boosting과 앙상블 모델에서 각각 72.77%, 72.37%까지 개선할 수 있었다.

```
Gradient Boosting Accuracy: 0.7277327935222672
              precision    recall  f1-score   support

           0       0.72      0.73      0.73       983
           1       0.73      0.72      0.73       993

    accuracy                           0.73      1976
   macro avg       0.73      0.73      0.73      1976
weighted avg       0.73      0.73      0.73      1976
```

```
Ensemble Model Accuracy: 0.7236842105263158
              precision    recall  f1-score   support

           0       0.72      0.73      0.72       983
           1       0.73      0.72      0.72       993

    accuracy                           0.72      1976
   macro avg       0.72      0.72      0.72      1976
weighted avg       0.72      0.72      0.72      1976
```

3. 결론 및 연구한계

(1) 연구한계 및 개선점

본 연구를 진행하며 마주한 연구한계와 개선점은 다음과 같다. 그 오브 레전드의 챔피언이나 플레이어의 특성에 대한 속성보다는 게임 내에서 얻어진 화폐와 경험치의 양에 집중하여 머신러닝을 진행했다. 리그 오브 레전드의 플레이 스타일이 다양하고 플레이어의 특성에 의해서 결정되는 요소가 다양하며 사용되는 챔피언의 포지션과 상대 챔피언과의 조합 등 다양한 변수가 게임의 승부에 영향을 미치기 때문에 사용한 데이터로 본 연구에선 얻어낸 72% 근방의 예측 성능을 얻기에는 어려움이 있다. 이를 보정하는 방법으로는 플레이어의 나이와 성별이나 성격과 같이 플레이어 중심의 속성들을 반영하고, 각 포지션과 챔피언 등 게임에 영향을 미치는 더 다양한 속성을 이용하여 머신러닝을 진행한다면 더욱 정확성 높은 예측이 가능할 것으로 보인다.

(2) 결론

학습 모델	Logistic Regression	Random Forest	Linear SVM	Gradient Boosting	Ensemble (voting)
정확도(%)	73.33	72.17	72.27	72.77	72.37

본 연구에서는 리그 오브 레전드의 다이아몬드 랭크에서 총 9879번의 게임 결과 중 7903번의 학습데이터를 기반으로 얻은 모델과 1976번의 test_data와 비교하여 최종 승리팀을 예측하였다. 머신러닝의 이진 분류 알고리즘으로 크게 Logistic Regression과 Random Forest, Linear Support Vector Machine 모델을 이용하였으며 이에 따라 나타나는 예측 성공률은 Logistic Regression은 73.33%, Random Forest의 경우

72.17%, Linear Support Vecter Machine의 경우 72.27%로 나타났다. 예측 성공률을 높이기 위한 방법으로 배깅을 활용하였고, Gradient Boosting을 활용하여 학습한 모델의 경우 72.77%를 보였으며 Gradient model과 Random Forest model을 배깅한 Ensemble 모델의 경우 72.37%로 소량 증가한 모습을 보였다. 또 ROC 곡선과 혼동 행렬을 출력하여 각각의 모델의 분류 성능을 시각화하여 분류 성능을 손쉽게 확인할 수 있도록 하였다. 추가적으로, 랜덤 포레스트 모델에서 특성 중요도를 파악하여 상대와의 골드/경험치 차이가 승부 결과에 지배적임을 알 수 있었으며 구체적인 게임 내에서는 전령이나 타워를 집중하기보다는 CS와 상대편을 처치하는데에 집중해야함을 수치적으로 확인하였다. 앞서 설명한 연구한계에서와 같이 롤은 10명이서 진행되는 다인의 상호작용이 일어나는 만큼, 게임 내의 데이터뿐만 아니라 플레이어의 특성과 챔피언 사이의 상호작용을 고려하여 머신러닝을 진행된다면 더욱 정확도 높은 완성된 연구를 진행할 수 있을 것이라 생각한다.

4. Appendix

4.1 용어사전

Warding totem: An item that a player can put on the map to reveal the nearby area. Very useful for map/objectives control.

Minions: NPC that belong to both teams. They give gold when killed by players.

Jungle minions: NPC that belong to NO TEAM. They give gold and buffs when killed by players.

Elite monsters: Monsters with high hp/damage that give a massive bonus (gold/XP/stats) when killed by a team.

Dragons: Elite monster which gives team bonus when killed. The 4th dragon killed by a team gives a massive stats bonus. The 5th dragon (Elder Dragon) offers a huge advantage to the team.

Herald: Elite monster which gives stats bonus when killed by the player. It helps to push a lane and destroys structures.

Towers: Structures you have to destroy to reach the enemy Nexus. They give gold.

Level: Champion level. Start at 1. Max is 18.

4.2 자료 출처

Dataset:

https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min/data