Second part: Data preparation + Modeling

Introduction

This report is related to Modeling and Evaluation parts for the [Cross-industry standard process for data mining](). We will test [Supervised learning]() models and compare each other using evaluation metrics.

A short description of the dataset:

The data set is related to the game called "League of Legends".The dataset contains the first 10 min of data in each game and also who won that game. Players have roughly the same level. There are 19 features per team (38 in total) collected after 10min in-game. This includes kills, deaths, gold, experience, level... The **target value is column `blueWins`**. A value of 1 means the blue team has won, 0 otherwise.

This data set have two types of **attributes**, the first group is the type **nominal**, concretely booleans (`blueWins`, `redFirstBlood`, etc.) and **numerical** type, mostly of them Integer, except the following ones, that are decimal values: - `redAvgLevel`, `redCSPerMin`, `redGoldPerMin` `blueAvgLevel`, `blueCSPerMin`, `blueGoldPerMin`.

Regarding to the **distribution** in this data set we found distributions as Logarithmic distribution, Beta distribution, Weibull_max distribution, Binomial distribution, but the most frequently is the **normal distribution**

For the other hand the **correlation** between attribute values is varying. Like **strong-positive** correlation (`blueKills - redDeads`, `redKills - blueDeads`, `blueDeads - redKills`, `redDeads - blueKills` and `blueTotalGold - blueGoldPerMin`) or **strong-negative** correlation (`blueFirstBlood - redFirstBlood`, `blueGoldDiff - redGoldDiff`, `blueExperienceDiff - redExperienceDiff`, `blueGoldDiff - redExperienceDiff`, `redGoldDiff - blueExperienceDiff`)

List of columns of our dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9879 entries, 0 to 9878
Data columns (total 40 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   gameId                        9879 non-null   int64
 1   blueWins                      9879 non-null   int64
 2   blueWardsPlaced               9879 non-null   int64
 3   blueWardsDestroyed            9879 non-null   int64
```

```
 4   blueFirstBlood                9879 non-null   int64
 5   blueKills                     9879 non-null   int64
 6   blueDeaths                    9879 non-null   int64
 7   blueAssists                   9879 non-null   int64
 8   blueEliteMonsters             9879 non-null   int64
 9   blueDragons                   9879 non-null   int64
 10  blueHeralds                   9879 non-null   int64
 11  blueTowersDestroyed           9879 non-null   int64
 12  blueTotalGold                 9879 non-null   int64
 13  blueAvgLevel                  9879 non-null   float64
 14  blueTotalExperience           9879 non-null   int64
 15  blueTotalMinionsKilled        9879 non-null   int64
 16  blueTotalJungleMinionsKilled  9879 non-null   int64
 17  blueGoldDiff                  9879 non-null   int64
 18  blueExperienceDiff            9879 non-null   int64
 19  blueCSPerMin                  9879 non-null   float64
 20  blueGoldPerMin                9879 non-null   float64
 21  redWardsPlaced                9879 non-null   int64
 22  redWardsDestroyed             9879 non-null   int64
 23  redFirstBlood                 9879 non-null   int64
 24  redKills                      9879 non-null   int64
 25  redDeaths                     9879 non-null   int64
 26  redAssists                    9879 non-null   int64
 27  redEliteMonsters              9879 non-null   int64
 28  redDragons                    9879 non-null   int64
 29  redHeralds                    9879 non-null   int64
 30  redTowersDestroyed            9879 non-null   int64
 31  redTotalGold                  9879 non-null   int64
 32  redAvgLevel                   9879 non-null   float64
 33  redTotalExperience            9879 non-null   int64
 34  redTotalMinionsKilled         9879 non-null   int64
 35  redTotalJungleMinionsKilled   9879 non-null   int64
 36  redGoldDiff                   9879 non-null   int64
 37  redExperienceDiff             9879 non-null   int64
 38  redCSPerMin                   9879 non-null   float64
 39  redGoldPerMin                 9879 non-null   float64
dtypes: float64(6), int64(34)
memory usage: 3.0 MB
```

A quick review of the chosen data mining goals:

The primary goal are prediction of the target column, in this case `blueWins`, with the use of some variables or fields in the database to predict unknown or future values of other variables of interest. Also, describing the data set as a whole by determining global characteristics and dividing examples into groups.

Discussion of the further steps:

The data mining task we are going to discuss is prediction(classification) of the data set. The probability of blue team winning is inversely correlated with red team, so our task will be analyse the blue team.

The modeling algorithms we are going to use for our dataset are **`Logistic Regression`**, **`K-Nearest Neighbours`** and **`Decision tree`**.

For the evaluation of the modeling algorithms we are going to use **`Cross-Validation`**.

Description of data preparation:

Our dataset does not have missing data so it's not necessary to assessing missing values. Also the all of our attributes are numerical, so the dataset is perfect to applicate the algorithm. We don't have to change the type of any attribute first.

Model creation:

First of all, we **remove** the **columns** that do not serve us. For example, attributes that brings nothing new to our dataset, that is, we can obtain that information from some other attribute. Some columns are repeated like:
`blueFirstblood/redFirstBlood`, `blueEliteMonster/redEliteMonster`
`blueDeath/redKills` etc.

Result of remove columns: *(See the code in appendix I.I)*

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9879 entries, 0 to 9878
Data columns (total 23 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   blueWins                     9879 non-null   int64
 1   blueWardsPlaced              9879 non-null   int64
 2   blueWardsDestroyed           9879 non-null   int64
 3   blueFirstBlood               9879 non-null   int64
 4   blueKills                    9879 non-null   int64
 5   blueDeaths                   9879 non-null   int64
 6   blueAssists                  9879 non-null   int64
 7   blueEliteMonsters            9879 non-null   int64
```

```
 8   blueDragons                   9879 non-null   int64
 9   blueHeralds                   9879 non-null   int64
 10  blueTowersDestroyed           9879 non-null   int64
 11  blueTotalGold                 9879 non-null   int64
 12  blueAvgLevel                  9879 non-null   float64
 13  blueTotalExperience           9879 non-null   int64
 14  blueTotalJungleMinionsKilled  9879 non-null   int64
 15  redWardsPlaced                9879 non-null   int64
 16  redWardsDestroyed             9879 non-null   int64
 17  redDeaths                     9879 non-null   int64
 18  redAssists                    9879 non-null   int64
 19  redTowersDestroyed            9879 non-null   int64
 20  redTotalGold                  9879 non-null   int64
 21  redAvgLevel                   9879 non-null   float64
 22  redTotalExperience            9879 non-null   int64
dtypes: float64(2), int64(21)
memory usage: 1.7 MB
```

Based on the modeling algorithm we choose. We had to clean the dataset to **avoid collinearity**. Also next, drop the columns that don't have strong correlation with `bluewin`. We will see above our final list of attributes.
(Code in appendix I.III)

	blueKills	blueDeaths	blueAssists	blueTotalGold	blueTotalExperience
0	9	6	11	17210	17039
1	5	5	5	14712	16265
2	7	11	4	16113	16221
3	4	5	5	15157	17954
4	6	6	6	16400	18543

There is a large variation of values in the variables within the data set. We must normalize these data. **Normalization** consists in compressing or extending the values of the variable to restrict the range of values.In this case, we are going to use Standard Scale.
The **StandardScaler** assumes your data is normally distributed within each feature and will scale them such that the distribution is now centred around 0, with a standard deviation of 1

Also, We divide the data set into two subsets: training set and test set. The **training set** is to fit the parameters and the **test set** is to assess the performance of the model.
In this case, we are going to divide our dataset in: 75% for training set and 25% for test set:

```
Train set: (7409, 5) (7409,)
Test set: (2470, 5) (2470,)
```
(Code in appendix I.III)

Applying Data Modeling algorithms:

Logistic Regression:

Logistic regression is a type of regression analysis used to predict the outcome of a categorical variable based on independent or predictive variables. The result is the impact of each variable on the odds ratio of the observed event of interest. The main advantage is to avoid confounding effects by analyzing the association of all variables together.

K-Nearest Neighbours:

The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. Also is a non-parametric algorithm

Decision tree.

Decision tree is used to build Classification Models. It builds classification models in the form of a tree-like structure, just like its name. This type of mining belongs to supervised class learning.

Decision Tree Algorithm Pseudocode:

1. Place the best attribute of the dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

Evaluation of the algorithms:

For evaluate the algorithms we are going to use `Cross-Validation`.
Cross-validation is a technique used to evaluate the results of a statistical analysis and verify that they are independent of the division between training and test data. Once the partition has been made, the model is trained once for each of the groups. Using all the groups except the one from the iteration to train and this one to validate the results.

To evaluate the algorithms we are going to use the class `gridSearchCV` in python. These class allows to evaluate and select the parameters of a model. By indicating a model and the parameters to be tested, `gridSearchCV` can evaluate the performance of the former based on the seconds by cross-validating.

To evaluate the algorithms we calculate these metrics:

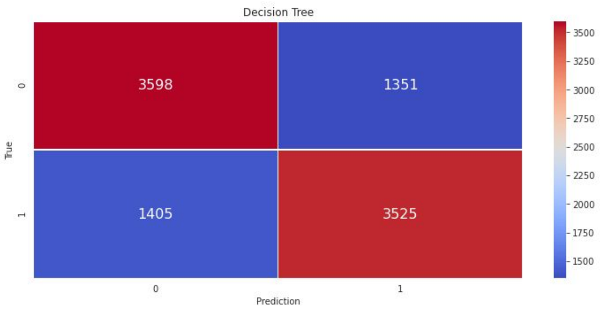
- **Accuracy,** it is the total percentage of items classified correctly. It is the most direct measure of the quality of the classifiers. It is a value between 0 and 1. The higher the better.
- **Recall,** is the fraction of the total amount of relevant instances that were actually retrieved.
- **Precision,** is the fraction of relevant instances among the retrieved instances.

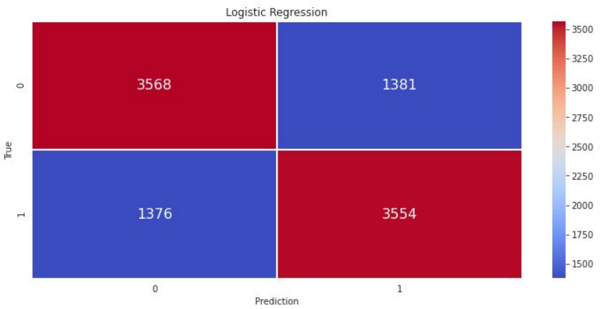
Results

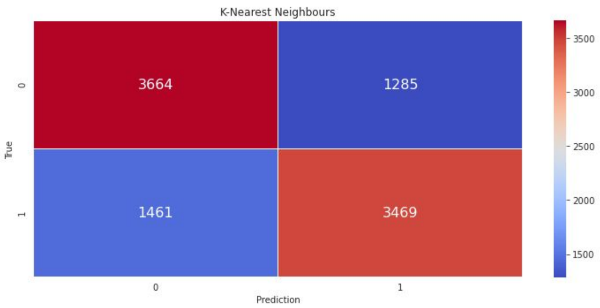
Metrics:

```
GridSearchCV Classifiers (OrderBy Accuracy)
+----------------------+----------+--------+----------+
|      Classifier      | Accuracy |  Recall | Precision |
+----------------------+----------+--------+----------+
| K-Nearest Neighbours |   72.2 %  | 70.37 % |  72.97 %  |
|    Decision Tree     |   72.1 %  |  71.5 % |  72.29 %  |
| Logistic Regression  |  72.09 %  | 72.09 % |  72.02 %  |
+----------------------+----------+--------+----------+
```

Confusion Matrix:

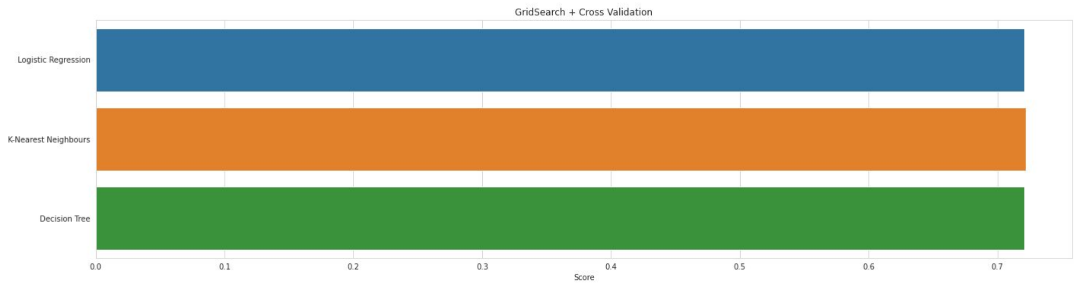






Conclusions:

As we can see, the results of scores are very similar, but we can say watching the confusion matrix that the results of KNN are the best predicting 0 when the result is 0 and also is the algorithm which less false-positives (predict 1 when finally will be 0).
We can say the opposite with Logistic Regression, that algorithm has the best score prediction 1-1 and also is the algorithm with less false false-negatives (predict 0 when finally will be 1).



Appendix I - Python's code used.

Appendix I.I Remove columns:

```
cols = ['gameId', 'redFirstBlood', 'redKills', 'redEliteMonsters',
'redDragons','redTotalMinionsKilled','redTotalJungleMinionsKilled',
       'redGoldDiff', 'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin',
'redHeralds','blueGoldDiff', 'blueExperienceDiff', 'blueCSPerMin',
       'blueGoldPerMin', 'blueTotalMinionsKilled']
#print(cols)
df_clean = df_clean.drop(labels='gameId', axis = 1)
df_clean = df_clean.drop(labels='redFirstBlood', axis = 1)
df_clean = df_clean.drop(labels='redKills', axis = 1)
df_clean = df_clean.drop(labels='redEliteMonsters', axis = 1)
df_clean = df_clean.drop(labels='redDragons', axis = 1)
df_clean = df_clean.drop(labels='redTotalMinionsKilled', axis = 1)
df_clean = df_clean.drop(labels='redTotalJungleMinionsKilled', axis = 1)
df_clean = df_clean.drop(labels='redGoldDiff', axis = 1)
df_clean = df_clean.drop(labels='redExperienceDiff', axis = 1)
df_clean = df_clean.drop(labels='redCSPerMin', axis = 1)
df_clean = df_clean.drop(labels='redGoldPerMin', axis = 1)
df_clean = df_clean.drop(labels='redHeralds', axis = 1)
df_clean = df_clean.drop(labels='blueGoldDiff', axis = 1)
df_clean = df_clean.drop(labels='blueExperienceDiff', axis = 1)
df_clean = df_clean.drop(labels='blueCSPerMin', axis = 1)
df_clean = df_clean.drop(labels='blueGoldPerMin', axis = 1)
df_clean = df_clean.drop(labels='blueTotalMinionsKilled', axis = 1)
```

Appendix I.II Avoid collinearity and columns that don't have strong correlation with bluewin.

```
# Based on the correlation matrix, let's clean the dataset again to
avoid collinearity
cols = ['blueAvgLevel', 'redWardsPlaced', 'redWardsDestroyed',
        'redDeaths', 'redAssists', 'redTowersDestroyed',
        'redTotalExperience', 'redTotalGold', 'redAvgLevel']
df_clean = df_clean.drop(cols, axis=1)

# Next we have to drop the columns with little correlation with bluewins
corr_list = df_clean[df_clean.columns[1:]].apply(lambda x:
x.corr(df_clean['blueWins']))
cols = []
for col in corr_list.index:
    if (corr_list[col]>0.23 or corr_list[col]<-0.23):
        cols.append(col)
```

Appendix I.III Normalize and divide into training set and test set

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score,
precision_score, confusion_matrix
from prettytable import PrettyTable

# Creamos la tabla que nos permitirá mostrar las métricas obtenidas.
metricas = PrettyTable()
metricas.field_names = ['Clasificador', 'Exactitud', 'Recall',
'Precisión']

# Guardaremos los resultados en un vector para ser mostrados en la
conclusión del trabajo.
resultados = []

X= df_clean

# Normalizamos los datos
X = StandardScaler().fit(X).transform(X)

# Asignamos los valores de entrenamiento y prueba
y = df['blueWins']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state = 28)
```

```
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Out[19]:
Train set: (7409, 5) (7409,)
Test set: (2470, 5) (2470,)
```

Appendix I.IV Cross-Validation with each Algorithm

```
from sklearn.linear_model import LogisticRegression

# Colocamos los valores de parámetros que queremos que GridSearchCV
pruebe por nosotros
grid_values = {
       'penalty': ['l1', 'l2'],
       'C':[.001,.009,0.01,.09,1,2,3,4,5,7,10,25],
       'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
        'fit_intercept' : [True, False]}

# Instanciamos la clase con los parámetros previamente asignados
grid_clf_acc = GridSearchCV(LogisticRegression(), param_grid =
grid_values, scoring = 'accuracy', verbose=False, n_jobs=-1)

# Seleccionamos la tabla entera, ya que el método se encargará de
realizar la técnica de Cross-Validation
grid_clf_acc.fit(X, y)

# Imprimimos los mejores parámetros seleccionados por GridSearchCV
print("Parámetros elegidos: " + str(grid_clf_acc.best_params_) + "\n")

# Predecimos los valores
y_pred_acc = grid_clf_acc.predict(X)

# Métricas de evaluación
exactitud = accuracy_score(y,y_pred_acc)
recall = recall_score(y,y_pred_acc)
precision = precision_score(y,y_pred_acc)
LR_confusion_matrix = confusion_matrix(y,y_pred_acc)

# Anexamos los resultados para ser mostrados posteriormente
resultados_grid_search.append(exactitud)

# Formateamos los datos para mostrarlos como % en la tabla.
exactitud = str(round(exactitud * 100, 2)) + " %"
recall = str(round(recall * 100, 2)) + " %"
precision = str(round(precision * 100, 2)) + " %"

metricas_grid_search.add_row(['Regresión Logística', exactitud, recall,
precision])
```

```
from sklearn.neighbors import KNeighborsClassifier


# Colocamos los valores de parámetros que queremos que GridSearchCV
pruebe por nosotros
grid_values = {"n_neighbors": [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 20, 22, 25, 30, 86, 87, 88, 100],
                 "weights": ["uniform","distance"],
                 "metric":["euclidean","manhattan"]}

# Instanciamos la clase con los parámetros previamente asignados
grid_clf_acc = GridSearchCV(KNeighborsClassifier(), param_grid =
grid_values, scoring = 'accuracy', verbose=False, n_jobs=-1)

# Seleccionamos la tabla entera, ya que el método se encargará de
realizar la técnica de Cross-Validation
grid_clf_acc.fit(X, y)

# Imprimimos los mejores parámetros seleccionados por GridSearchCV
print("Parámetros elegidos: " + str(grid_clf_acc.best_params_) + "\n")

# Predecimos los valores
y_pred_acc = grid_clf_acc.predict(X)

# Métricas de evaluación
exactitud = accuracy_score(y,y_pred_acc)
recall = recall_score(y,y_pred_acc)
precision = precision_score(y,y_pred_acc)
KNN_confusion_matrix = confusion_matrix(y,y_pred_acc)

# Anexamos los resultados para ser mostrados posteriormente
resultados_grid_search.append(exactitud)

# Formateamos los datos para mostrarlos como % en la tabla.
exactitud = str(round(exactitud * 100, 2)) + " %"
recall = str(round(recall * 100, 2)) + " %"
precision = str(round(precision * 100, 2)) + " %"

metricas_grid_search.add_row(['K-Nearest Neighbours', exactitud, recall,
precision])
```

```
from sklearn.tree import DecisionTreeClassifier


# Colocamos los valores de parámetros que queremos que GridSearchCV
pruebe por nosotros
grid_values = {'max_depth': np.arange(1, 21),
               'min_samples_leaf': [1, 5, 10, 20, 50, 100]}

# Instanciamos la clase con los parámetros previamente asignados
grid_clf_acc = GridSearchCV(DecisionTreeClassifier(), param_grid =
grid_values, scoring = 'accuracy', verbose=False, n_jobs=-1)

# Seleccionamos la tabla entera, ya que el método se encargará de
realizar la técnica de Cross-Validation
grid_clf_acc.fit(X, y)

# Imprimimos los mejores parámetros seleccionados por GridSearchCV
print("Parámetros elegidos: " + str(grid_clf_acc.best_params_) + "\n")

# Predecimos los valores
y_pred_acc = grid_clf_acc.predict(X)

# Métricas de evaluación
exactitud = accuracy_score(y,y_pred_acc)
recall = recall_score(y,y_pred_acc)
precision = precision_score(y,y_pred_acc)
DT_confusion_matrix = confusion_matrix(y,y_pred_acc)

# Anexamos los resultados para ser mostrados posteriormente
resultados_grid_search.append(exactitud)

# Formateamos los datos para mostrarlos como % en la tabla.
exactitud = str(round(exactitud * 100, 2)) + " %"
recall = str(round(recall * 100, 2)) + " %"
precision = str(round(precision * 100, 2)) + " %"

metricas_grid_search.add_row(['Decision Tree', exactitud, recall,
precision])
```

Bibliography / Webography

League of Legends Diamond Ranked Games (10 min)
Data ScienceTutorial for Beginners
Machine Learning Tutorial for Beginners
How to choose the best K in KNN (K nearest neighbour) classification
Why and how to Cross Validate a Model?
sklearn.tree.DecisionTreeClassifier — scikit-learn 0.23.1 documentation
sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.23.1 documentation
sklearn.linear_model.LogisticRegression — scikit-learn 0.23.1 documentation
3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.23.1 documentation
https://github.com/jbofill10/LoL-MatchOutcome-Predictor
k vecinos más próximos
Trabajo Practico Nº1 - IA
Precauciones a la hora de normalizar datos en Data Science
How To Implement The Decision Tree Algorithm From Scratch In Python
Cross-industry standard process for data mining
Supervised learning
4.2 Logistic Regression | Interpretable Machine Learning
Understanding Logistic Regression
Machine Learning Basics with the K-Nearest Neighbors Algorithm
Cross Validation
About Feature Scaling and Normalization
How Decision Tree Algorithm works
Decision Tree Algorithm — Explained

Data Mining Lecture: Basic Problems and Definitions.
Data Mining Lecture: Classification and regression. Part 1.
Data Mining Lecture: Classification and regression. Part 2.
Data Mining Lecture: Classification and regression. Part 3.
Data Mining Lecture: Evaluating the results. Part 1.
Data Mining Lecture: Evaluating the results. Part 2.
Data Mining Lecture: More advanced input and output.