Predicting CS:GO match winner after the first half of the game

1 Introduction

Counter Strike: Global Offensive (CS:GO) is an online multiplayer game where players are divided into two sides: Terrorists (t) and Counter Terrorists (ct). The teams compete against each other in different maps i.e., places where the matches take place. In a competitive match type, which we are interested in, a match is won by winning 16 individual rounds before opponent reaches 15 rounds. In case of a tie (15-15) match is decided in overtime. The sides (t, ct) the teams play on are switched after 15 rounds of the match is played. Thus, a match consists of a first and a second half.

The application domain is prediction of the match winner after the first half of the match is played using machine learning. The second section discusses the used dataset, data points in the dataset and problem description. The third section discusses feature selection process, machine learning methods used and design of training and validation sets. The fourth section discusses and compares the results obtained from using the methods. Finally, the fifth section concludes the report.

2 Problem formulation

2.1 Data

The dataset (from Kaggle [1]) contains data from professional CS:GO matches ranging from year 2015 to 2020. One data point represents one professional CS:GO match results. The uploader of the dataset has scraped the data from hltv.org – a website for CS:GO match schedules and statistics. The dataset contains 45773 matches without missing features or labels. Columns included in the data points are date, team names, map name, starting side, number of round wins for each team in each side, match id number, team ranks and winner of the match. The types of data in the columns include continuous (time data) categorical (teams, map), numerical (scores, ids) and binary (match winner, starting side).

To minimize outliers and get better results, the data will be limited to matches from 2018 to 2020 and will only include matches with both teams having at least rank 100. The limitations are made because the maps and playstyle change over time and the lesser ranked matches aren't as predictable. After the limitations the dataset still has 16400 data points.

2.2 Problem description

The goal is to predict outcome of a CS:GO match based on the first half's results and team performance. The idea is to utilize supervised learning and try to use first half results of the match and team performance related data as features and the match outcome (winner team 1 / team 2 (binary)) as label.

3 Methods

3.1 Features

Since the objective is to predict the match outcome only after the first half of the match, we can use the first half results as features for this problem. When predicting the match outcome mid-game, it is important to know how much the other team is leading the game. Thus, the score difference after the first half (team 1 rounds won – team 2 rounds won) is used as a feature.

Other important factors contributing to the final match result include team rank (skill difference), map and starting side. Map is important because teams perform differently on different maps. Starting side is important because on one side it is usually easier to win rounds than on the other. The fact that which starting side is easier also depends on the map.

The map and starting side need to be transformed into some type of numerical value, so that it can be used as inputs in the model. I calculated for each map the "ct" side win percent, in other words, the percent represents the number of times a team won a half on "ct" side divided by the total number of halves played on the map. In addition to "ct" win percent, the starting side is used as a feature.

The used features for the model are: first half round difference (numerical), rank difference (numerical), starting side (binary) and map "ct" side win percentage (numerical between 0-1).

3.2 Logistic regression

Since the label of the problem is binary, it is natural to use a binary classification method. Thus, one binary classification method I am going to use is logistic regression. Logistic regression allows the classification of data points into two categories. A common label space for binary labels is $\{0, 1\}$, but we will use $\{1, 2\}$ since it is optional. Logistic regression uses linear maps $h(x) = w^T x$ as hypothesis space. [2] Linear maps are used because they are simple and easy to understand, and it can be intuitively deduced that the features e.g., first half result has a quite linear relationship to match result.

Logistic regression typically uses logistic loss as loss function to measure how well the linear hypothesis performs [2]. Also, logistic loss is readily implemented in the sklearn library [3]. That is why I am going to use logistic loss as the loss function for this model.

Logistic regression requires features to be standardized. Thus, the features were scaled with sklearn StandardScaler before training the model.

3.3 SVC

The second machine learning method used is support vector classifier, which is a classification method. SVC uses the same hypothesis space as Logistic Regression discussed in Section 3.2, thus same reasoning applies to why SVC is used for the problem. I use the same labels space $\{1, 2\}$ for SVC as with logistic regression. With the use of SVC, we will see which linear map method performs better.

Hinge loss is used for SVC as the loss function because it is common for SVC and readily available in the sklearn library [4].

3.4 Train-test split

Choosing the split is usually done by trial and error [2]. Common train test splits discussed in course seem to range from 60-80 percent to training and rest to test and validation. In addition, our dataset is quite large, and the effect of outliers should be low. Also, the use of methods such as k-fold cross-validation require more computation on the already large dataset. Because of these factors, I simply split ("single split") the data to 60 percent training and 20 percent to both testing and validation.

4 Results

Table 1 shows the accuracies of logistic regression and SVC and the errors of their loss functions logistic loss and hinge loss. The training and validation accuracies are very similar for both models at 77-78%. The difference for the models seems to be in that errors for SVC seem to be higher. Also, the difference between training error and validation error seems to be bigger in SVC. These would indicate that logistic regression would be more stable method for our problem. So, I select logistic regression as the model.

Model	SVC	Logistic regression
Training accuracy	0.7801829268292683	0.7724593495934959
Validation accuracy	0.774390243902439	0.7701219512195122
Training error	1.1589430894308943	0.47128576867447647
Validation error	1.1817073170731707	0.47321091159625306
Test error	1.1655487804878049	0.47928018783664683

Table 1: accuracies and errors

The design of the test set was explained in Section 3.4. From Table 1 it is seen that the test error for logistic regression is nearly the same (0.47-0.48) as validation error and training error.

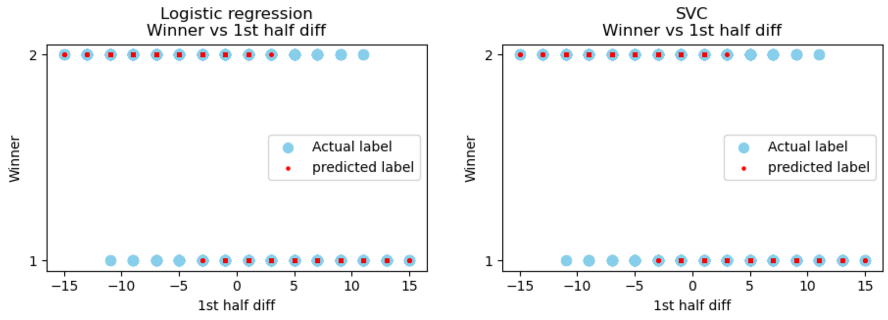


Figure 1: winner label vs 1st half diff for both models

5. Conclusion

In this report I used classification methods namely logistic regression and SVC to predict the winner of a CS:GO match after the first half. I discussed the dataset and data points, selected features by domain knowledge, used single split (60/20/20) for the construction of testing and validation sets and applied the models using the sklearn library. The results show that linear regression would be better method for the problem at least with current features. Logistic regression was chosen because it had less difference between errors and accuracies. If e.g., training error would be much lower than validation error, it could indicate overfitting in the model. The very small difference between logistic regression training and validation errors could indicate that the model will generalize well to new data.

The accuracy of the logistic regression model is pretty good at 77% considering the problem at hand. However, there is certainly room for improvement. From Figure 1 we can see that neither of the models were able to predict wins of the teams that had been first down five or more rounds after the first half (i.e., the teams that made a “comeback”). This indicates that our model is quite closely following the results of the first half. The model still however manages to predict wins even when a team is losing by four rounds.

To further improve the model in the future, good starting point would probably be to add more features. With more significant features it could be possible to predict the “comeback” situations from Figure 1 better. When including many features, it could also be a good idea to try different models like decision trees or neural networks.

References

[1] https://www.kaggle.com/datasets/mateusdmachado/csgo-professional-matches

[2] https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

[3] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.hinge_loss.html

Appendix

ML_project

October 2, 2023

[47]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, log_loss, confusion_matrix,␣
 ↪hinge_loss, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC


df_results = pd.read_csv('results.csv')
df_results.drop(columns=['match_winner', 'event_id', 'map_wins_1',␣
 ↪'map_wins_2'], inplace=True)
df_results = df_results.loc[df_results['_map'] != 'Default']
df_results = df_results.loc[df_results['date'] > '2018-01-01']
df_results = df_results.loc[(df_results['rank_1'] <= 100) &␣
 ↪(df_results['rank_2'] <= 100)]
df_results = df_results.reset_index()
df_results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16400 entries, 0 to 16399
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   index        16400 non-null  int64
 1   date         16400 non-null  object
 2   team_1       16400 non-null  object
 3   team_2       16400 non-null  object
 4   _map         16400 non-null  object
 5   result_1     16400 non-null  int64
 6   result_2     16400 non-null  int64
 7   map_winner   16400 non-null  int64
 8   starting_ct  16400 non-null  int64
 9   ct_1          16400 non-null  int64
 10  t_2           16400 non-null  int64
 11  t_1           16400 non-null  int64
 12  ct_2          16400 non-null  int64
 13  match_id      16400 non-null  int64
 14  rank_1        16400 non-null  int64
 15  rank_2        16400 non-null  int64
dtypes: int64(12), object(4)
memory usage: 2.0+ MB
```

```
[48]: df_results['1st_half_winner'] = df_results.apply(lambda df: 1 if␣
      ↪(df['starting_ct'] == 1 and df['ct_1'] > df['t_2']) or (df['starting_ct'] ==␣
      ↪2 and df['t_1'] > df['ct_2']) else 2, axis = 1)
      df_results['1st_half_diff'] = df_results.apply(lambda df: (df['ct_1'] -␣
      ↪df['t_2']) if (df['starting_ct'] == 1) else (df['t_1'] - df['ct_2']), axis =␣
      ↪1)
      df_results['rank_diff'] = (df_results['rank_2'] - df_results['rank_1'])
      df_results['map_diff'] = df_results['result_1'] - df_results['result_2']

      maps = df_results['_map'].unique()
      ct_wins_maps = {}
      ct = {}
      for _map in maps:
          ct_r = df_results[df_results['_map'] == _map]['ct_1'].sum() +␣
      ↪df_results[df_results['_map'] == _map]['ct_2'].sum()
          t_r = df_results[df_results['_map'] == _map]['t_1'].sum() +␣
      ↪df_results[df_results['_map'] == _map]['t_2'].sum()
          ct_wins_maps[_map] = ct_r/(t_r + ct_r)

          ct_half_won = df_results[(df_results['_map'] == _map) &␣
      ↪((df_results['ct_1'] > df_results['t_2']))].shape[0]
          ct_half_won += df_results[(df_results['_map'] == _map) &␣
      ↪((df_results['ct_2'] > df_results['t_1']))].shape[0]
          ct_half_won += df_results[(df_results['_map'] == _map) &␣
      ↪((df_results['ct_2'] > df_results['t_1']) & ((df_results['ct_1'] >␣
      ↪df_results['t_2'])))].shape[0]
          n_halfs = df_results[(df_results['_map'] == _map)].shape[0] * 2
          ct[_map] = ct_half_won/n_halfs


      df_results['map_ct_win_percent'] = df_results.apply(lambda row:␣
      ↪ct[row['_map']], axis=1)

      corr_df = df_results.drop(columns=['date'])

      corr_df['_map'] = corr_df['_map'].astype('category').cat.codes
```

```
team_to_num = {}
team_to_comebacks = {}
team_to_secured = {}
for index, team in enumerate(pd.concat([corr_df['team_1'], corr_df['team_2']]).
 ↪unique()):
    team_to_num[team] = index
    comeback_games = corr_df[(corr_df['team_1'] == team) &␣
 ↪(corr_df['1st_half_diff'] <= -1)].shape[0] + \
        corr_df[(corr_df['team_2'] == team) & (corr_df['1st_half_diff'] >= 1)].
 ↪shape[0]

    secured_games = corr_df[(corr_df['team_1'] == team) &␣
 ↪(corr_df['1st_half_diff'] >= 1)].shape[0] + \
        corr_df[(corr_df['team_2'] == team) & (corr_df['1st_half_diff'] <= -1)].
 ↪shape[0]

    comebacks = corr_df[(corr_df['team_1'] == team) & (corr_df['1st_half_diff']␣
 ↪<= -1) & (corr_df['map_winner'] == 1)].shape[0] + \
        corr_df[(corr_df['team_2'] == team) & (corr_df['1st_half_diff'] >= 1) &␣
 ↪(corr_df['map_winner'] == 2)].shape[0]

    secured = corr_df[(corr_df['team_1'] == team) & (corr_df['1st_half_diff']␣
 ↪>= 1) & (corr_df['map_winner'] == 1)].shape[0] + \
        corr_df[(corr_df['team_2'] == team) & (corr_df['1st_half_diff'] <= -1)␣
 ↪& (corr_df['map_winner'] == 2)].shape[0]

    comeback_percent = comebacks / comeback_games if comeback_games != 0 else 0
    secured_percent = secured / secured_games if secured_games != 0 else 0

    team_to_comebacks[team] = comeback_percent
    team_to_secured[team] = secured_percent
    #print(secured_percent)



corr_df['1st_loser_comeback'] = corr_df.apply(lambda row:␣
 ↪team_to_comebacks[row['team_1']] if row['1st_half_winner'] == 2 else␣
 ↪team_to_comebacks[row['team_2']], axis=1)
corr_df['1st_winner_secure'] = corr_df.apply(lambda row:␣
 ↪team_to_secured[row['team_1']] if row['1st_half_winner'] == 1 else␣
 ↪team_to_secured[row['team_2']], axis=1)
corr_df['team_1_comeback'] = corr_df['team_1'].apply(lambda row:␣
 ↪team_to_comebacks[row])
corr_df['team_2_comeback'] = corr_df['team_2'].apply(lambda row:␣
 ↪team_to_comebacks[row])
```

```
corr_df['team_1_secured'] = corr_df['team_1'].apply(lambda row:␣
 ↪team_to_secured[row])
corr_df['team_2_secured'] = corr_df['team_2'].apply(lambda row:␣
 ↪team_to_secured[row])
corr_df['team_1'] = corr_df['team_1'].apply(lambda row: team_to_num[row])
corr_df['team_2'] = corr_df['team_2'].apply(lambda row: team_to_num[row])


df_results['1st_loser_comeback'] = df_results.apply(lambda row:␣
 ↪team_to_comebacks[row['team_1']] if row['1st_half_winner'] == 2 else␣
 ↪team_to_comebacks[row['team_2']], axis=1)
df_results['1st_winner_secure'] = df_results.apply(lambda row:␣
 ↪team_to_secured[row['team_1']] if row['1st_half_winner'] == 1 else␣
 ↪team_to_secured[row['team_2']], axis=1)
df_results['team_1_comeback'] = df_results['team_1'].apply(lambda row:␣
 ↪team_to_comebacks[row])
df_results['team_2_comeback'] = df_results['team_2'].apply(lambda row:␣
 ↪team_to_comebacks[row])
df_results['team_1_secured'] = df_results['team_1'].apply(lambda row:␣
 ↪team_to_secured[row])
df_results['team_2_secured'] = df_results['team_2'].apply(lambda row:␣
 ↪team_to_secured[row])
df_results['team_1'] = df_results['team_1'].apply(lambda row: team_to_num[row])
df_results['team_2'] = df_results['team_2'].apply(lambda row: team_to_num[row])

df_results.drop(columns=['result_1', 'result_2', 'date', 'ct_1', 'ct_2', 't_2',␣
 ↪'t_1']).head(5)
```

[48]:

```
   index  team_1  team_2      _map  map_winner  starting_ct  match_id  rank_1  \
0      0       0     147     Dust2           2            2   2340454      62
1      1       0     147   Inferno           2            2   2340454      62
2      3       1      20   Inferno           2            2   2340453      61
3      4       1      20   Vertigo           2            2   2340453      61
4      5       2      18  Overpass           2            2   2340456      71

   rank_2  1st_half_winner  1st_half_diff  rank_diff  map_diff  \
0      63                2            -15          1       -16
1      63                2             -5          1        -3
2      38                2             -1        -23        -9
3      38                2             -7        -23        -8
4      41                2             -5        -30        -3

   map_ct_win_percent  1st_loser_comeback  1st_winner_secure  team_1_comeback  \
0            0.531675            0.333333           0.716418         0.333333
1            0.543625            0.333333           0.716418         0.333333
2            0.543625            0.230769           0.829787         0.230769
```

```
3            0.531325           0.230769           0.829787           0.230769
4            0.676056           0.243094           0.772727           0.243094

   team_2_comeback  team_1_secured  team_2_secured
0         0.238806        1.000000        0.716418
1         0.238806        1.000000        0.716418
2         0.250000        0.466667        0.829787
3         0.250000        0.466667        0.829787
4         0.169231        0.711765        0.772727
```

[49]:
```
#testing
#fig = plt.figure()
#axis = fig.add_subplot(1, 1, 1)

#axis.scatter(df_results[df_results['map_winner'] == 2]['1st_half_diff'],␣
 ↪df_results[df_results['map_winner'] == 2]['rank_diff'], s=10, c='b',␣
 ↪marker="s", label='first', alpha=0.3)
#axis.scatter(df_results[df_results['map_winner'] == 1]['1st_half_diff'],␣
 ↪df_results[df_results['map_winner'] == 1]['rank_diff'], s=10, c='r',␣
 ↪marker="o", label='second', alpha=0.3)
```

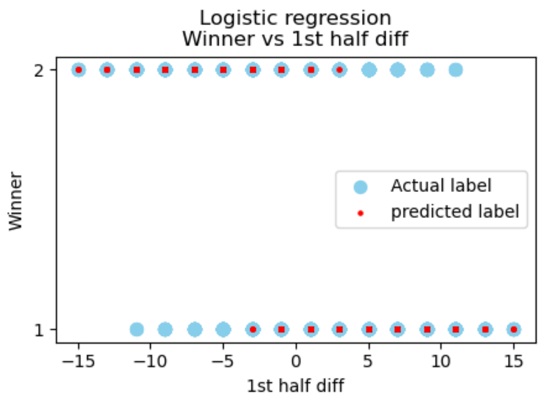
[50]:
```
# , 'team_1', 'team_2', 'team_2_comeback', '1st_loser_comeback',␣
 ↪'1st_winner_secure', 'team_1_comeback', 'team_2_secured', 'team_1_secured'
X = df_results[['1st_half_diff', 'rank_diff', 'starting_ct',␣
 ↪'map_ct_win_percent']].to_numpy().reshape(-1, 4)
y = df_results['map_winner'].to_numpy()
#X = corr_df.drop(columns=['map_winner', 'map_diff', 'result_1', 'result_2',␣
 ↪'t_1', 't_2', 'ct_2', 'ct_1']).to_numpy().reshape(-1, 12)




X_train_, X_test_, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=45)
X_train_, X_val_, y_train, y_val = train_test_split(X_train_, y_train,␣
 ↪test_size=0.25, random_state=45)

scale = StandardScaler()
X_train = scale.fit_transform(X_train_)
X_test = scale.transform(X_test_)
X_val = scale.transform(X_val_)


model = LogisticRegression(penalty="l2", C=1000)
model_2 = SVC(C=10, kernel='rbf')

# ---- FIT MODELS ----
```

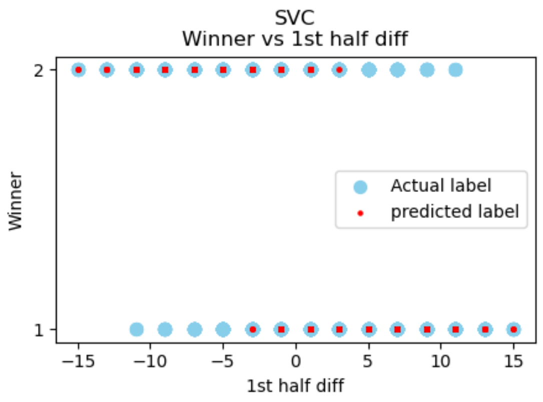
```
model.fit(X_train, y_train)
model_2.fit(X_train, y_train)

# ---- PREDICT ----
y_pred_train = model.predict(X_train)
accuracy_train = accuracy_score(y_train, y_pred_train)
y_pred_val = model.predict(X_val)
accuracy_val = accuracy_score(y_val, y_pred_val)
y_pred_train_2 = model_2.predict(X_train)
accuracy_train_2 = accuracy_score(y_train, y_pred_train_2)
y_pred_test_2 = model_2.predict(X_test)
y_pred_val_2 = model_2.predict(X_val)
accuracy_val_2 = accuracy_score(y_val, y_pred_val_2)

# ---- Errors ----
train_err = log_loss(y_train, model.predict_proba(X_train))
val_err = log_loss(y_val, model.predict_proba(X_val))
test_err = log_loss(y_test, model.predict_proba(X_test))

train_err_2 = hinge_loss(y_train, y_pred_train_2)
val_err_2 = hinge_loss(y_val, y_pred_val_2)
test_err_2 = hinge_loss(y_test, y_pred_test_2)

conf_mat = confusion_matrix(y_val, y_pred_val)
conf_mat_2 = confusion_matrix(y_val, y_pred_val_2)
print("---- LOGISTIC REGRESSION ----")
print(f"Training accuracy logRes: {accuracy_train}")
print(f"Validation accuracy logRes: {accuracy_val}")

print(f"Training error logLoss: {train_err}")
print(f"Validation error logLoss: {val_err}")
print(f"Test error logLoss: {test_err}")
print(f"confusion matrix\n{conf_mat}")
print("\n")
print("---- SVC ----")
print(f"Training accuracy SVC: {accuracy_train_2}")
print(f"Validation accuracy SVC: {accuracy_val_2}")

print(f"Training error hingeLoss: {train_err_2}")
print(f"Validation error hingeLoss: {val_err_2}")
print(f"Test error hingeLoss: {test_err_2}")
print(f"confusion matrix\n{conf_mat_2}")
```

```
---- LOGISTIC REGRESSION ----
Training accuracy logRes: 0.7724593495934959
Validation accuracy logRes: 0.7701219512195122
Training error logLoss: 0.47128576867447647
```

```
Validation error logLoss: 0.47321091159625306
Test error logLoss: 0.47928018783664683
confusion matrix
[[1374  377]
 [ 377 1152]]


---- SVC ----
Training accuracy SVC: 0.7801829268292683
Validation accuracy SVC: 0.774390243902439
Training error hingeLoss: 1.1589430894308943
Validation error hingeLoss: 1.1817073170731707
Test error hingeLoss: 1.1655487804878049
confusion matrix
[[1377  374]
 [ 366 1163]]
```

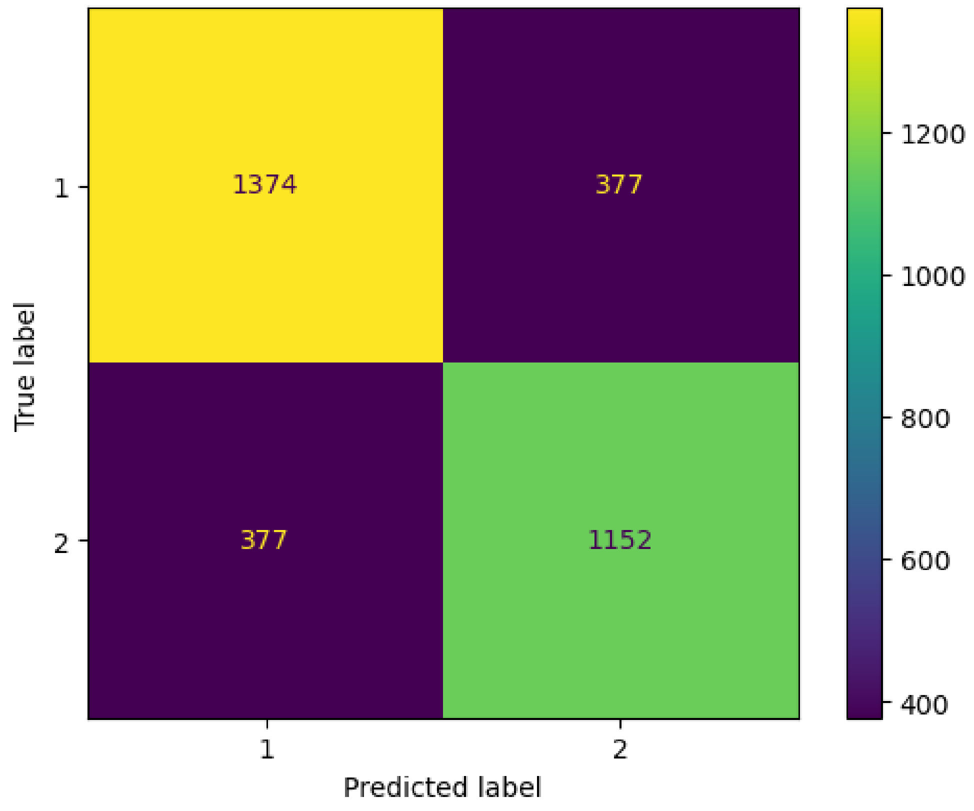
[51]:
```
fig, ax = plt.subplots(1, 2, figsize=(11,3))
ax[0].set_xlabel("1st half diff")
ax[0].set_yticks([1, 2])
ax[0].set_ylabel('Winner')
ax[0].set_title("Logistic regression\nWinner vs 1st half diff")
ax[0].scatter([x[0] for x in X_val_], y_val,s=50,c="skyblue",label="Actual␣
 ↪label")
ax[0].scatter([x[0] for x in X_val_], y_pred_val,color='r',s=5,label='predicted␣
 ↪label')
ax[0].legend()
ax[1].set_xlabel("1st half diff")
ax[1].set_yticks([1, 2])
ax[1].set_ylabel('Winner')
ax[1].set_title("SVC\nWinner vs 1st half diff")
ax[1].scatter([x[0] for x in X_val_], y_val,s=50,c="skyblue",label="Actual␣
 ↪label")
ax[1].scatter([x[0] for x in X_val_],␣
 ↪y_pred_val_2,color='r',s=5,label='predicted label')
ax[1].legend()
plt.show()
```





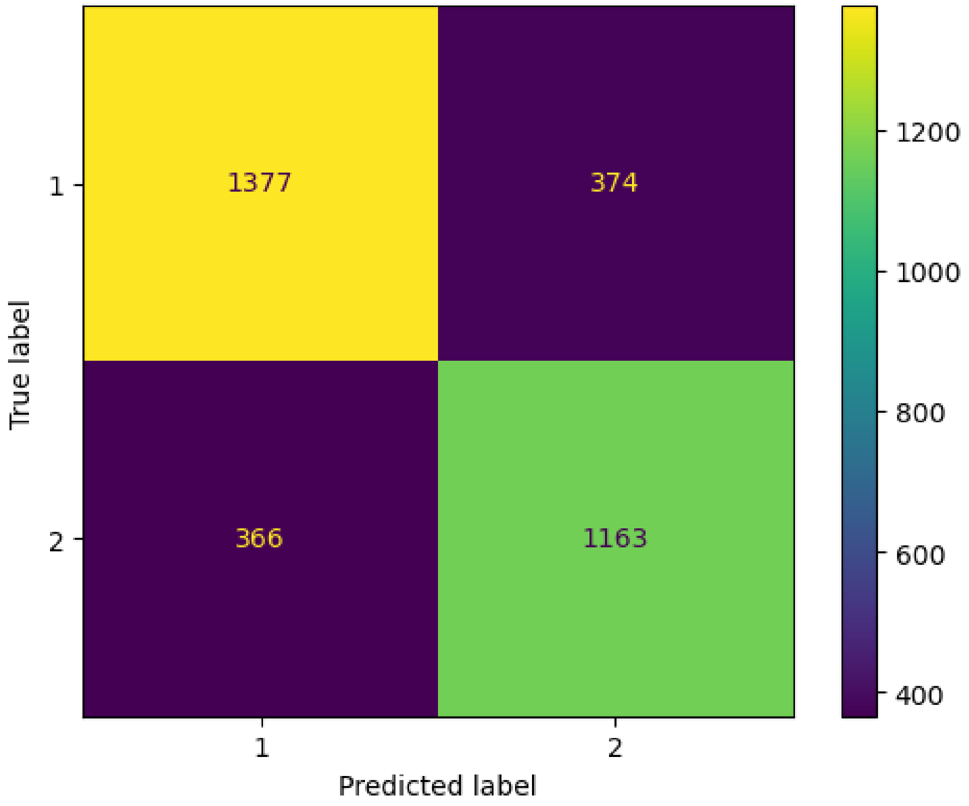
[52]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat,
                              display_labels=model.classes_)
disp.plot()
```

[52]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f0b10a75cf0>
```



```
[53]: disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat_2,
                                     display_labels=model_2.classes_)
      disp.plot()
```

```
[53]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
      0x7f0b10454f40>
```



```
[ ]:
```