ML Report

Introduction

Mixed martial arts, also known as MMA, is a full-contact combat sport that is based on various (mixed) combat sport elements, such as striking, grappling and ground fighting. [1] Ultimate Fighting Championship, also known as UFC, is an MMA promotion company based in the USA, from which various fighting events have been produced. [2]

In a varied combat sport such as MMA, it is possible to win in many different ways. Possible outcomes of a fight include a knockout or technical knockout (KO / TKO), a submission, a point-based decision (unanimous, majority, split, technical etc.), or a disqualification / no contest / forfeit situation. [3]

In this report, I will be attempting to classify the outcomes of UFC fights into technical knockouts and submissions based on the winning fighter's data. The problem formulation section goes into more detail about the specifics of the problem, the methods section discusses the ML models chosen and other design choices. The results section gives context to the results obtained from the models and loss functions. Finally, everything is summarised and elaborated on in the conclusion section.

Problem formulation

The dataset was obtained from Kaggle [4]. The dataset includes over 13 000 data points, each representing a UFC fighter's statistics from a specific fight.

The features used are takedown attempts and accuracy, total strike attempts and accuracy and submission attempts (no submission accuracy is taken into account, since if a submission succeeds, the fight ends). The accuracy features are calculated based on the attacks landed over the attack attempts (e.g. total strikes landed / total strikes attempted). [4] All of the features are numerical, since they are made up of integers and floating points (number of strikes attempted (int) , average accuracy of said strikes (float) etc.). Since our main interest is

classifying the method of winning, our label is the outcome method of the fight ‘method’. It is binary, since there are only two possible outcomes - the T/KO or the submission.

Methods

The dataset displays winning and losing fighters from UFC fights, so there are two data points per fight. We’ll only be looking at the winning fighters’ statistics. This means that from 13000 data points we have about 6500 separate fighters to analyse. For simplicity’s sake, we will only be looking at fights that ended in T/KO or a submission, which means we have 3515 data points left to look at.

The preprocessing done to achieve the 3515 data points is shown in the appendix (appendix B). We must, however, note, that when simplifying this model to only look at T/KOs and submissions instead of all possible outcomes to fights, we must assume that all data put through the model in the future will have the same two winning methods as the only winning method options. The reason for the simplification is that decision-based wins are based on points gained from techniques, and for that it would be necessary to look at the other fighter’s data as well. Even then, the features might not correlate enough to be able to accurately classify the fights.

The features I chose at the end are based on the visualisations I did on the data that was left (appendix A). I was originally going to include knockdowns, height differences, significant strikes and reversals, but each of these features either had no correlation or were irrelevant to the result, were already included in other features (significant strikes are part of total strikes) or there was too much missing data for it to be included (height and reach differences). The features also make sense intuitively - one would assume that a fight won by a submission would have more submission attempts, and a fight won by a knockout would have a lot of strike attempts and possibly a high strike accuracy.

For the ML methods, supervised learning will be used since the data is labelled. Due to the nature of the binary classification problem, logistic regression is a simple method to use as a starting point. It should work well with the linear features of the data, since it uses the same hypothesis space as linear regression.

For the second method I will be using SVC since it is effective in high-dimensional spaces and therefore good when using multiple variables. It is also more effective in handling imbalanced datasets, which is good since about a third of the data points are submission outcomes and two thirds are T/KO-outcomes.

Logistic loss and hinge loss will be chosen as loss functions for linear regression and SVC respectively, because they align with the optimisation objectives and principles of both ML methods, and are the most commonly used ones. In addition to this, both loss functions are already implemented in a library, which makes their use easier.

Regardless of the aforementioned slightly imbalanced data, random sampling will be used in the training and validation set construction due to the difference being relatively small. There is a fairly large amount of data points, so a 60/20/20 training, testing and validation split should suffice. The sets are constructed using train_test_split from sklearn.model_selection.

Results

	Logistic Regression (log loss)	SVC (hinge loss)
Training error	2.7430091839513056	0.4359886201991465
Testing error	2.3584751862011224	0.4423897581792319
Validation error	2.4097463859011468	0.47083926031294454
Accuracy score	0.9331436699857752	0.930298719772404

As seen from the above table, the accuracies are very similar in both the LR and SVC models. However, there are some differences in the errors. With LR, the training error is larger than the validation error, whereas with SVC it is the other way around. This could mean that the SVC model has slight overfitting issues, whereas the LR model might have some underfitting issues. This means that LR might be too simple to capture all the underlying patterns in the data. All in all both of these methods seem to be performing very similarly. This is most likely due to the

nature of the data and how both methods are based on linearity. The errors of the methods cannot be directly compared to each other due to the losses being calculated using different functions.

Based on all the above information I will be choosing SVC as my final method, since it is more complex and thus could be more accurate in the future when the possible overfitting is dealt with. The test set was constructed using the same sklearn train_test_split method as mentioned before. The final test error was approximately 0.44.

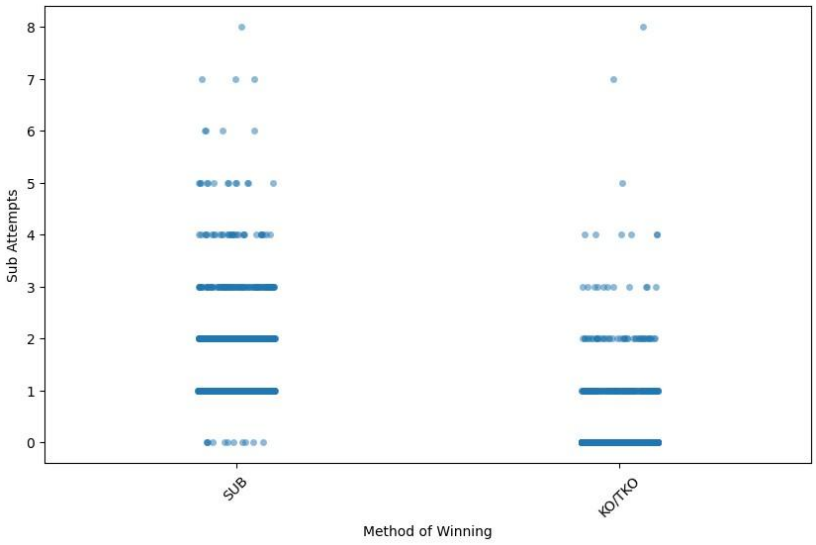
Conclusion

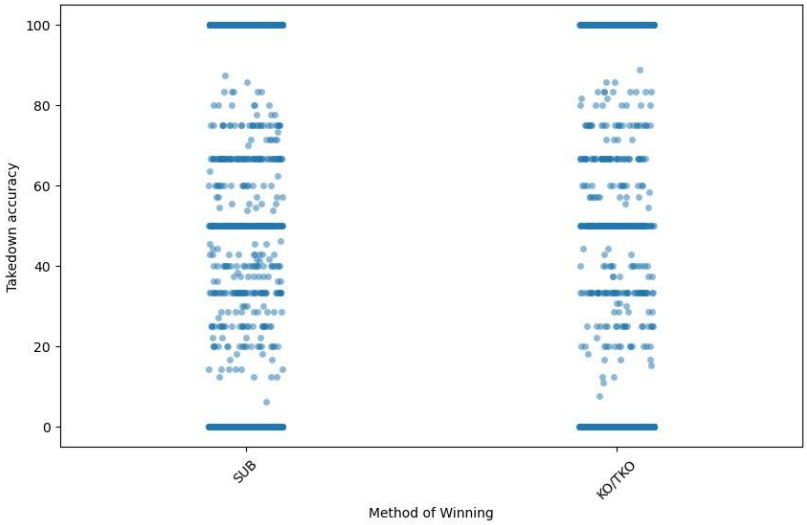
All in all, when trying to classify UFC fights based on their outcome, it seems that both logistic regression and SVC could work rather well. However, the final result of a 93% accuracy is not optimal, since ideally we'd like to get as close to 100% as possible. This could be achieved by changing the ML methods used or by fixing over/ -underfitting issues. Another option would be to add more features, and take into account the losing fighter's tactics as well, or even just fighter statistics, such as height or reach difference. However, most likely the data used simply isn't "perfect" enough and has a fair amount of outliers, since many things can affect the outcome of a fight, not just the accuracy of takedown or strike attempts.

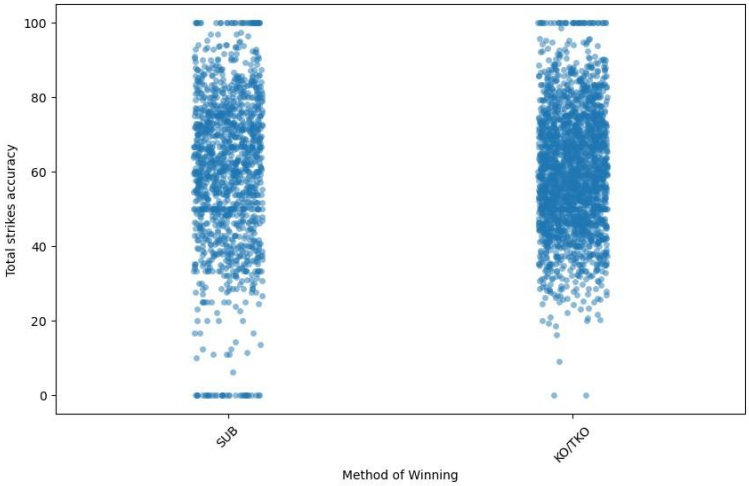
REFERENCES & BIBLIOGRAPHY

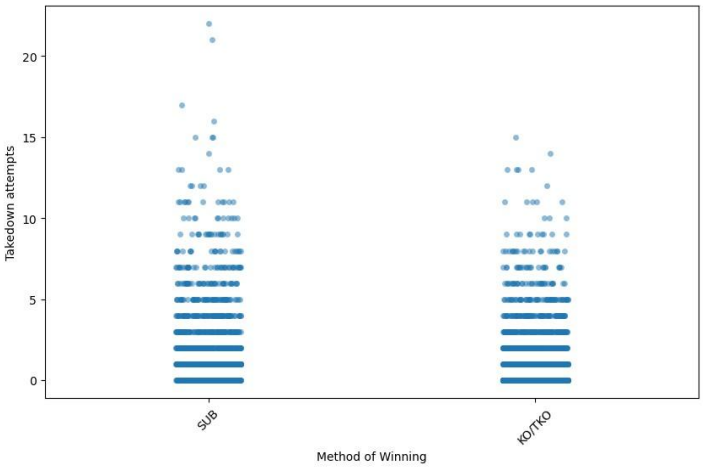
https://en.wikipedia.org/wiki/Mixed_martial_arts [1]
https://en.wikipedia.org/wiki/Ultimate_Fighting_Championship [2]
ChatGPT prompt: "What are the different ways a UFC fight can end?" [3]
https://chat.openai.com/ (ChatGPT 3.5)
Kaggle dataset:
https://www.kaggle.com/datasets/danmcinerney/mma-differentials-and-elo [4]
Original UFC stats website:
ufcstats.com (not secure)
A. Jung,"Machine Learning: The Basics," Springer, Singapore, 2022 [5]

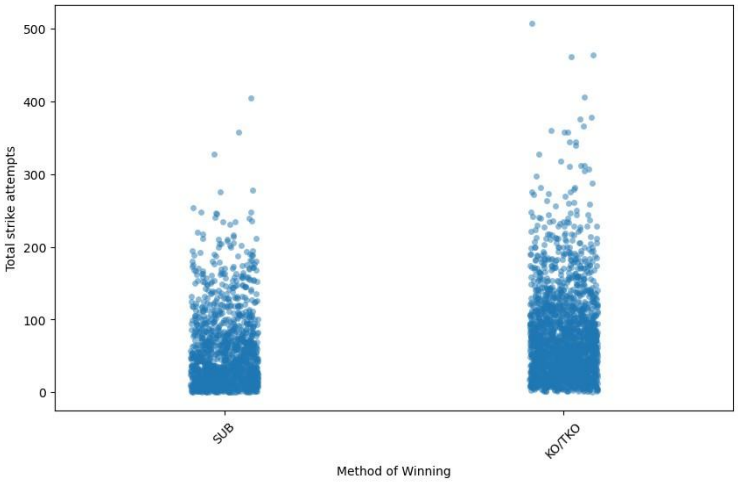
APPENDIX A: VISUALISATIONS











APPENDIX B: CODE

MLProject2

October 11, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score,␣
 ↪classification_report, log_loss, hinge_loss, confusion_matrix,␣
 ↪ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import KFold, cross_val_score

df = pd.read_csv('masterdataframe.csv') #reading the file

# transform sub into 0, KO into 1
df['method'].replace({"SUB": 0, "KO/TKO": 1}, inplace = True)

#removing most of the columns and leaving specific ones
df = df[['result', 'method', 'sub_attempts', 'takedowns_attempts',
         'total_strikes_attempts', 'takedowns_accuracy',␣
 ↪'total_strikes_accuracy']]

#only looking at KO or SUB outcomes
win = df.loc[(df['result'] == 1) & df['method'].isin([0,1])] #only look at␣
 ↪winners and situations ending in a TKO/KO or submission



#checking for null values
win.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3515 entries, 1 to 13319
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
```

```
 0   result                  3515 non-null   int64
 1   method                  3515 non-null   object
 2   sub_attempts            3515 non-null   int64
 3   takedowns_attempts      3515 non-null   int64
 4   total_strikes_attempts  3515 non-null   int64
 5   takedowns_accuracy      3515 non-null   float64
 6   total_strikes_accuracy  3515 non-null   float64
dtypes: float64(2), int64(4), object(1)
memory usage: 219.7+ KB
```

[2]:
```
# check for imbalances in number of datapoints
countko = win['method'].value_counts()[1]
countsub = win['method'].value_counts()[0]

print(f"The number of KOs is {countko} and Subs is {countsub}")
```

```
The number of KOs is 2193 and Subs is 1322
```

[3]:
```
# set features and labels
y = win[['method']].astype(int)
X = win[['takedowns_attempts', 'takedowns_accuracy', 'total_strikes_attempts',
         'total_strikes_accuracy', 'sub_attempts']]
```

[4]:
```
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

[5]:
```
# training the data for logistic regression
lr = LogisticRegression()
lr.fit(X_train, y_train.values.ravel())
```

[5]:
```
LogisticRegression()
```

[6]:
```
svc = SVC(kernel = 'linear')
svc.fit(X_train, y_train.values.ravel())
```

[6]:
```
SVC(kernel='linear')
```

[7]:
```
# training data predictions
y_pred_lr = lr.predict(X_train)
y_pred_svc = svc.predict(X_train)
```

[8]:
```
# testing data predictions
y_pred_test_lr = lr.predict(X_test)
```

```
y_pred_test_svc = svc.predict(X_test)
```

[9]:
```
# validation data predictions
y_pred_val_lr = lr.predict(X_val)
y_pred_val_svc = svc.predict(X_val)
```

[10]:
```
# check the errors of the different subsets
error_lr = log_loss(y_train, y_pred_lr)
error_svc = hinge_loss(y_train, y_pred_svc)

error_test_lr = log_loss(y_test, y_pred_test_lr)
error_test_svc = hinge_loss(y_test, y_pred_test_svc)

error_val_lr = log_loss(y_val, y_pred_val_lr)
error_val_svc = hinge_loss(y_val, y_pred_val_svc)

print('Logistic Regression')
print('the training error was:', error_lr)
print('the testing error was:', error_test_lr)
print('the validation error was:', error_val_lr)

print('SVC')
print('the training error was:', error_svc)
print('the testing error was:', error_test_svc)
print('the validation error was:', error_val_svc)
```

```
Logistic Regression
the training error was: 2.7430091839513056
the testing error was: 2.4097463859011468
the validation error was: 2.4097463859011468
SVC
the training error was: 0.4359886201991465
the testing error was: 0.47083926031294454
the validation error was: 0.47083926031294454
```

[11]:
```
acc_lr = accuracy_score(y_val, y_pred_val_lr)
acc_svc = accuracy_score(y_val, y_pred_val_svc)



print(acc_lr)
print(acc_svc)
```

```
0.9331436699857752
0.930298719772404
```

[12]:
```
# k-fold cross-validation for logistic regression
k_folds = KFold(n_splits = 5)
scores_lr = cross_val_score(lr, X, y.values.ravel(), cv = k_folds)
print(scores_lr.mean())
```

0.923470839260313

[13]:
```
# k-fold cross-validation for svc
k_folds = KFold(n_splits = 5)
scores_svc = cross_val_score(svc, X, y.values.ravel(), cv = k_folds)
print(scores_svc.mean())
```

0.9331436699857752

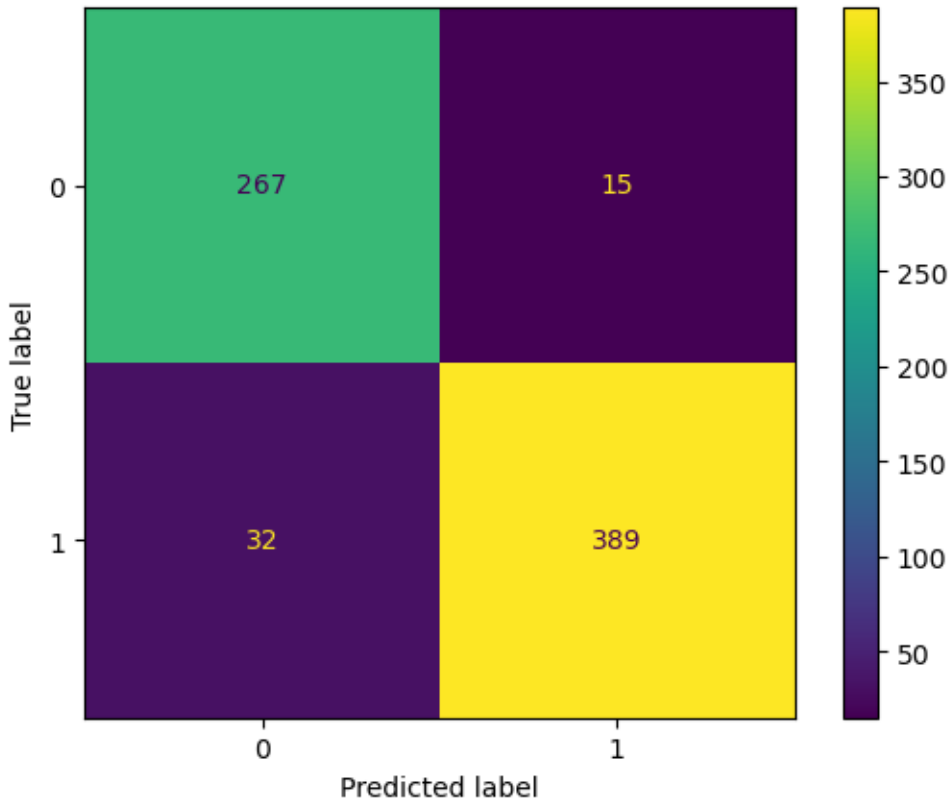
[14]:
```
# confusion matrix test for logistic regression

cm = confusion_matrix(y_test, y_pred_test_lr)

disp = ConfusionMatrixDisplay(confusion_matrix = cm)

disp.plot()
```

[14]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f38ba8027d0>



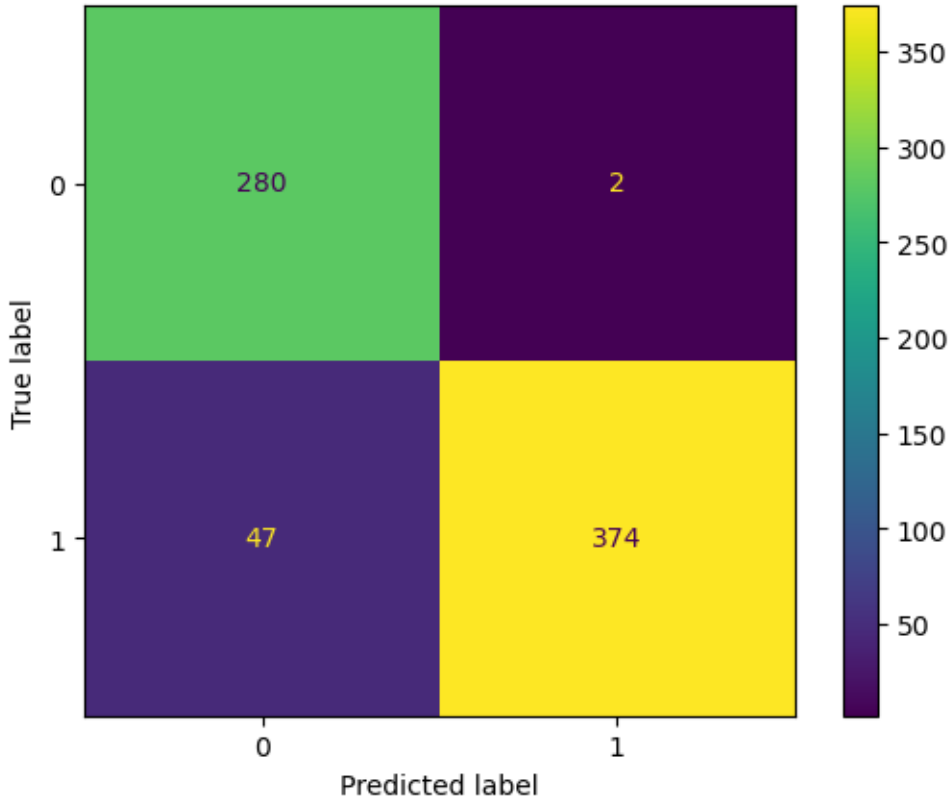
[15]:
```
# confusion matrix test for SVC

cm = confusion_matrix(y_test, y_pred_test_svc)

disp = ConfusionMatrixDisplay(confusion_matrix = cm)

disp.plot()
```

[15]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f385973be50>
```

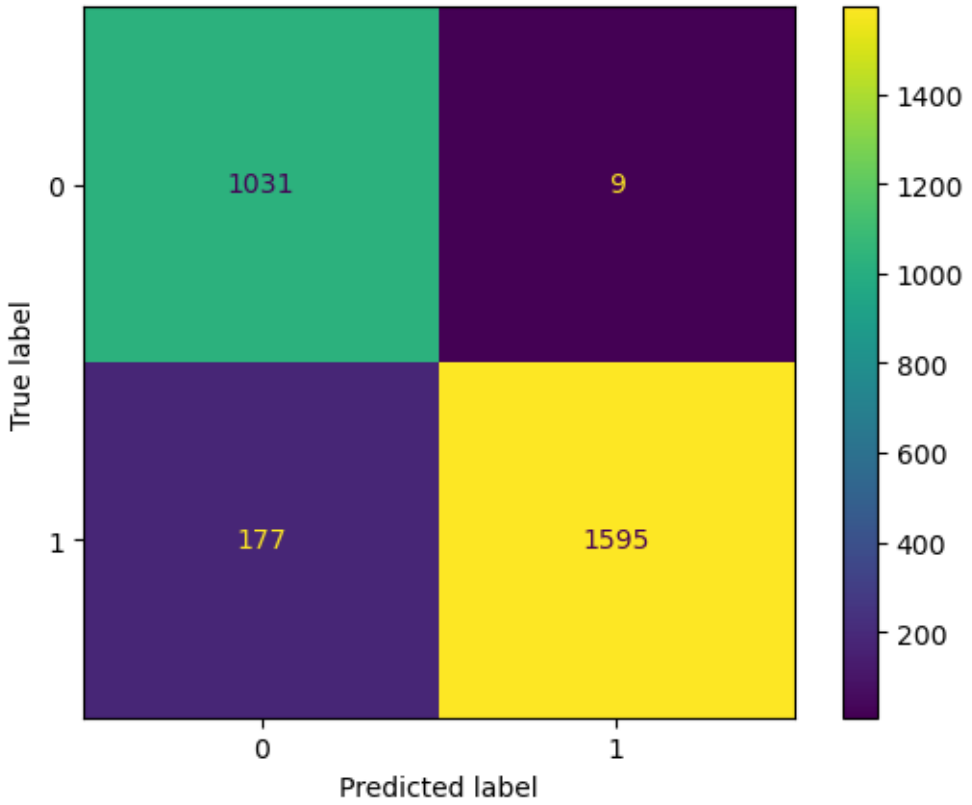


[17]:
```
cm = confusion_matrix(y_train, y_pred_svc)

disp = ConfusionMatrixDisplay(confusion_matrix = cm)

disp.plot()
```

[17]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f38902a0730>
```


1031
9
177
1595
0
1
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200

[ ]: