

Machine-Learning

August 25, 2025

1 Machine Learning project in SoSe 2025 at HTW Saar

1.1 Idea

The goal of this project is predicting the genre(s) of a game/bundle through its given description(s)

1.2 Dataset

For our project we use a Steam Dataset provided on moodle, since it has all information we plan on using. The Dataset has been cut to only 2000 data points to be runnable on weaker devices.

```
[1]: import numpy as np
import pandas as pd
from sklearn import set_config

set_config(transform_output="pandas")

dataset = pd.read_csv("./games_march2025_cleaned_2k.csv", sep=",")
print(dataset.head(1))
```

	appid	name	release_date	required_age	price	dlc_count	\
0	730	Counter-Strike 2	2012-08-21	0	0.0	1	
							detailed_description \
0							For over two decades, Counter-Strike has offer...
							about_the_game \
0							For over two decades, Counter-Strike has offer...
							short_description reviews ... \
0							For over two decades, Counter-Strike has offer... NaN ...
							average_playtime_2weeks median_playtime_forever median_playtime_2weeks \
0							879 5174 350
							discount peak_ccu tags \
0							0 1212356 {'FPS': 90857, 'Shooter': 65397, 'Multiplayer'...
							pct_pos_total num_reviews_total pct_pos_recent num_reviews_recent

0

86

8632939

82

96473

[1 rows x 47 columns]

1.3 Preparation of the Dataset

1.3.1 Removing Uniques

We would remove the following features from the Training-Set as they can/could uniquely identify a datapoint, but we don't as they will be removed in the next step anyway - AppId - Name of the Game - Release Date - Reviews - Header Image - Website - Support URL - Support Email - MetaCritic URL - Developer - Publisher - Screenshots - Movies - Estimated Owners

```
[2]: #dataset.drop(['appid', 'name', 'release_date', 'reviews', 'header_image',
↳ 'website', 'support_url', 'support_email', 'metacritic_url', 'notes',
↳ 'developers', 'publishers', 'screenshots', 'movies', 'estimated_owners'],
↳ axis=1, inplace=True)
#print(dataset.head())
```

1.4 Hold onto necessary information

Our model should turn a textual description of a game into its genre. For that we need all the textual information a game has, as well as the genres of the game. We use a ColumnTransformer to drop all unnecessary lines, merge all descriptions of a game into one big description and hold onto the genres

It is important to use `verbose_feature_names_out=False` so the feature names don't get changed

```
[3]: from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer

# desc, genres
column_transformer = ColumnTransformer([
    # merge all descriptions
    ('desc', FunctionTransformer(lambda X: X.fillna('').agg(' '.join,
↳ axis=1).to_frame(name="desc")),
    [
        ('detailed_description', 'about_the_game', 'short_description'),
        ('pass', 'passthrough', ['genres']),
    ],
    verbose_feature_names_out=False
)
dataset = column_transformer.fit_transform(dataset)
print(dataset.head())
```

desc \

```
0 For over two decades, Counter-Strike has offer...
1 LAND, LOOT, SURVIVE! Play PUBG: BATTLEGROUNDS ...
2 The most-played game on Steam. Every day, mill...
3 When a young street hustler, a retired bank ro...
4 Edition Comparison Ultimate Edition The Tom Cl...
```

```

                                genres
0                                ['Action', 'Free To Play']
1  ['Action', 'Adventure', 'Massively Multiplayer...
2                                ['Action', 'Strategy', 'Free To Play']
3                                ['Action', 'Adventure']
4                                ['Action']

```

1.4.1 Adding missing Information

Some Games might not have any descriptions. For these we Input an Empty String.

```

[4]: # missing numeric values => mean
dataset.fillna(dataset.mean(numeric_only=True), inplace=True)
# missing strings => empty string?
dataset.fillna('', inplace=True)
# drop all lines with missing values
dataset.dropna(inplace=True)

```

1.5 Transform Genres

The genre information currently is a string holding a python array of genres. While this is machine-readable, we need One-Hot-Encoding for our model to work.

Serializing the String-Array The “ast” library can interpret python strings as python code, and as such will be used for serializing the genres.

```

[5]: import ast

dataset['genres'] = dataset['genres'].map(lambda s: ast.literal_eval(s))
print(dataset['genres'].head())

```

```

0                                [Action, Free To Play]
1  [Action, Adventure, Massively Multiplayer, Fre...
2                                [Action, Strategy, Free To Play]
3                                [Action, Adventure]
4                                [Action]
Name: genres, dtype: object

```

One-Hot-Encoding an Python-Array The sklearn `OneHotEncoder()` is only able to work with an 1D Array of different classes, such as `['Politics', 'Sport', 'Culture']`. Every datapoint can only have one concurrent classification. Steam allows an app/bundle to have multiple genres. As such, our dataset has an 2D Array of different classes, which sklearn’s `MultiLabelBinarizer()` does support.

```

[6]: from sklearn.preprocessing import MultiLabelBinarizer

mlb_genres = MultiLabelBinarizer()
genres_encoded = mlb_genres.fit_transform(dataset.pop('genres'))

```

```
genres_df = pd.DataFrame(genres_encoded, columns=mlb_genres.classes_)
print(genres_df.head())
```

	Action	Adventure	Casual	Early Access	Free To Play	Gore	Indie	\
0	1	0	0	0	1	0	0	
1	1	1	0	0	1	0	0	
2	1	0	0	0	1	0	0	
3	1	1	0	0	0	0	0	
4	1	0	0	0	0	0	0	

	Massively Multiplayer	RPG	Racing	Simulation	Sports	Strategy	Violent
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

With this, our target matrix is completed.

1.5.1 Structurizing Text

If we want our Model to be able to use text as an input, we have to vectorize the text. TF-IDF (Inverse Document Frequency) is an easy way of transforming each word into a feature with a 0 to 1 value.

```
[7]: from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(dataset['desc']) # matrix, not pandas df
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.
    ↳get_feature_names_out())
print(tfidf_df.head())
```

	00	000	000km	000th	00am	00f	00i	00p	00v	01	...	\	
0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
1	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
2	0.0	0.0	0.0	0.162349	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
3	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
4	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[5 rows x 29056 columns]

With this our feature matrix is completed

```
[8]: X = tfidf_df
     y = genres_df
```

1.6 The Model

Removing unpredicatble Datapoints Some genres have too little datapoints to be predictable. The 10k Dataset has 14 Classes that have less than 10 Datapoints, usually only 1 to 4. These have too big of a probability that they will fall into only the train or test data and therefore will be removed.

```
[9]: # remove genres that have less than min_entries entries -> probability of
     ↪broken split to big
mask = (y == 1).sum() >= 10
print("Before" + str(y.shape))
y_prep = y.loc[:, mask]
print("After" + str(y_prep.shape))
```

Before(1999, 14)

After(1999, 12)

Some Datapoints don't have a genre assigned (all feature values in y are 0, either from the start or after we removed them one step before). The model we use can't handle such cases, thus they have to be removed. We filter after all values that we can use with a mask, and apply that mask to our matrices.

```
[10]: mask = y.sum(axis=1).map(lambda x: x > 0)
      print((mask == False).sum()) # count of unpredictable datapoints

      X_clean = X[mask]
      y_clean = y_prep[mask]
```

13

2 Splitting up data

We have to split up our data into training and testing data. Using random_state=0 guarantees reproducibility.

```
[11]: from sklearn.model_selection import train_test_split

      X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean,
      ↪random_state=0)
```

We also do a little cleanup session before proceeding.

```
[12]: import gc

      # Initial dataset loading
      del dataset
```

```

del column_transformer

# preparation of y
del mlb_genres
del genres_encoded
del genres_df

# preparation of X
del tfidf_df
del tfidf_matrix

# Initial Dataset
del X
del y
# Removing Genres with less than 5 datapoints
del y_prep

# Sorting out dead datapoints (all target values are 0)
del X_clean
del y_clean
del mask

gc.collect()

```

[12]: 82

Now that all data is prepared, we need to choose a Classification Model that meets our stanadrds.

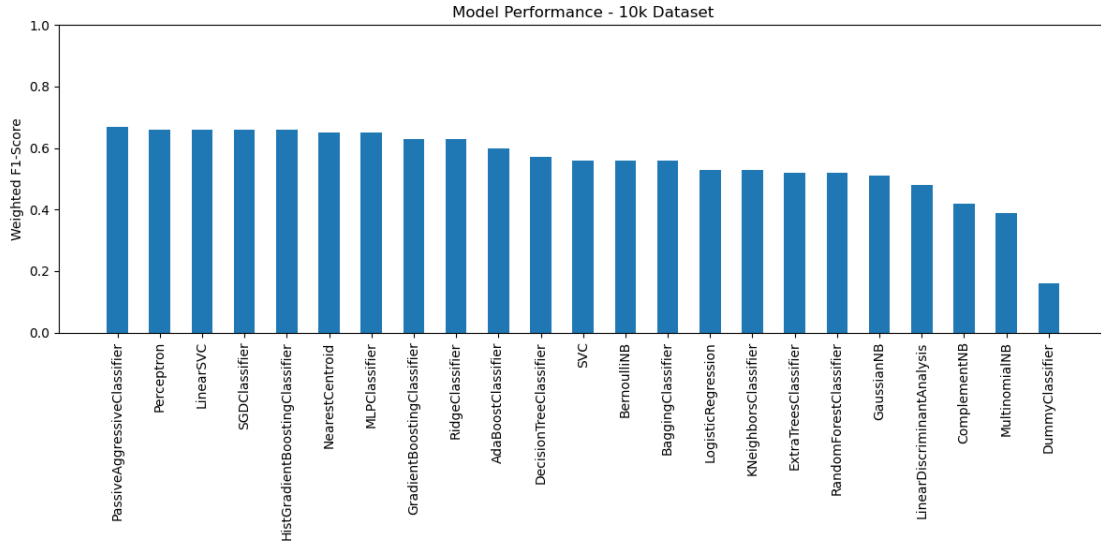
3 Excursion: Choosing a classification Model

`sklearn` has many different classification Models to choose from, but we only have limited time and computing power. As such, we tested many different models on the small dataset and chose the best performing ones for the big dataset.

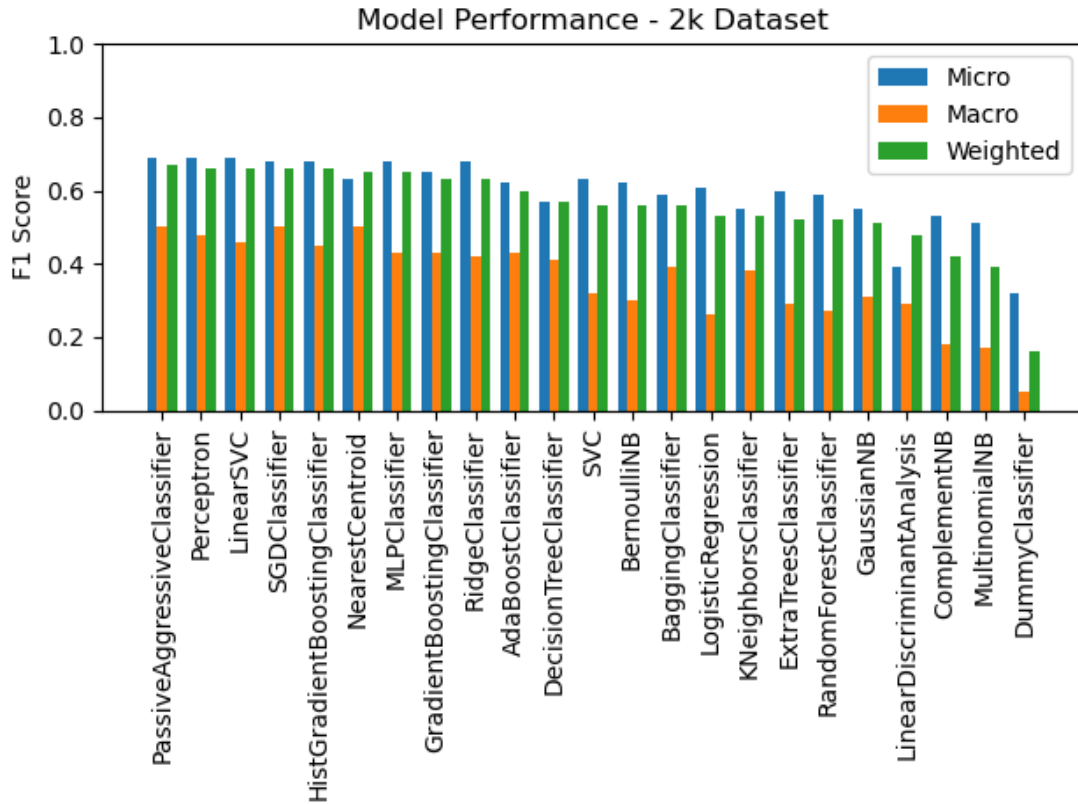
3.0.1 Initial Comparison

We won't put the comparison script in this notebook, but you can find it in the `compare_models_2k.py` file and try it out yourself. There were some rules as a baseline for comparison: - All Hyperparameters are set to default - All iteration limits are set to 3000 (exception: `MLPClassifier` with 300, where i-limit are epochs instead of iterations) - All `random_states` are set to 0

Running all models with that configuration yields the following weighted F1-Scores (results as seen in the `games_march2025_cleaned_2k_i3k` folder):



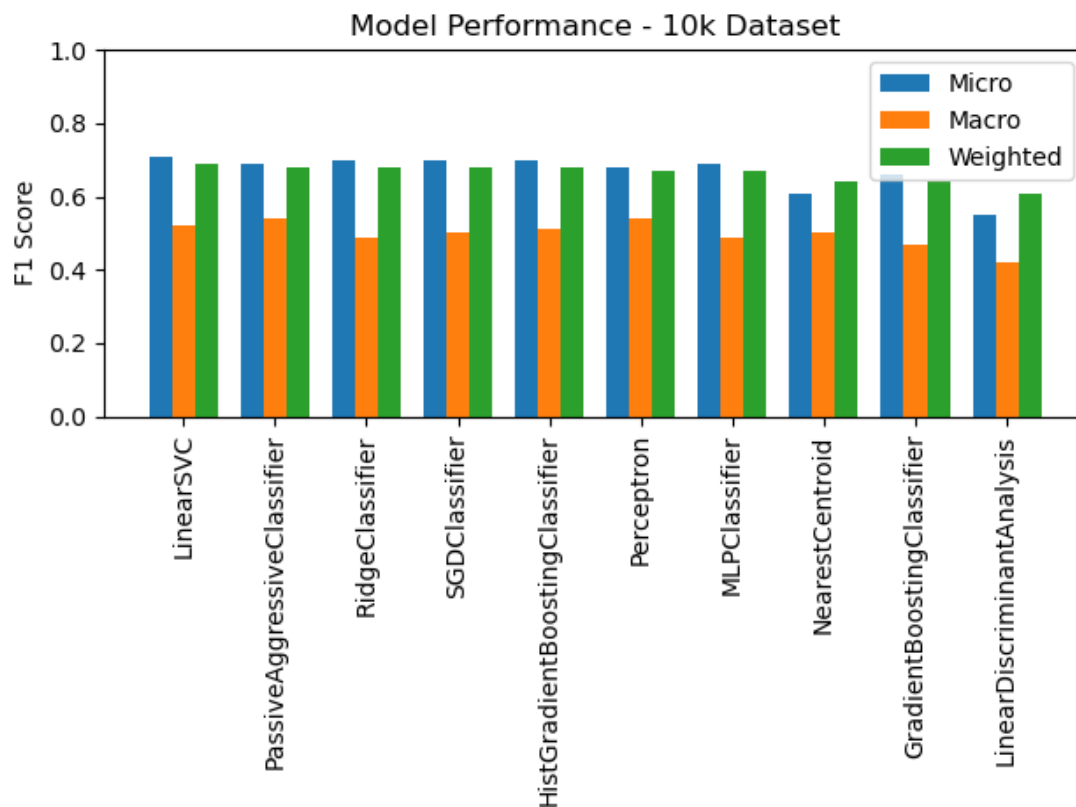
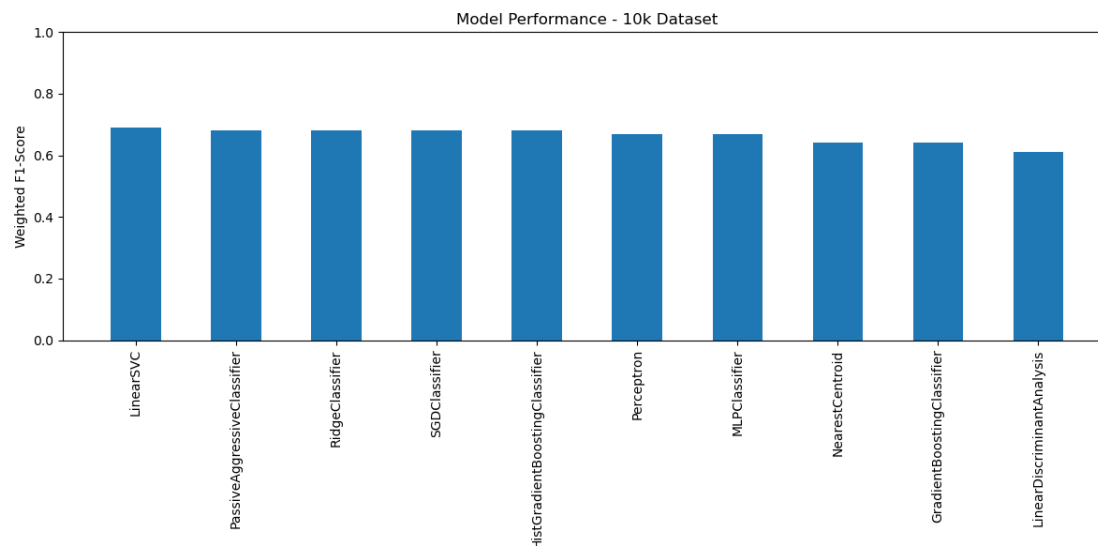
If we also compare Micro/Macro values, we see that all models have a much lower Macro-F1 than Micro/Weighted-F1. That is because the Dataset does not contain enough datapoints for every class (test data for 2 classes is 0 in the 2k dataset), so we should proceed to the 10k Dataset.



The 10 best performing models which will run on the 10k Dataset with the same rules as before:
 1. PassiveAggressiveClassifier 2. Perceptron 3. LinearSVC 4. SGDClassifier 5. HistGradientBoost-

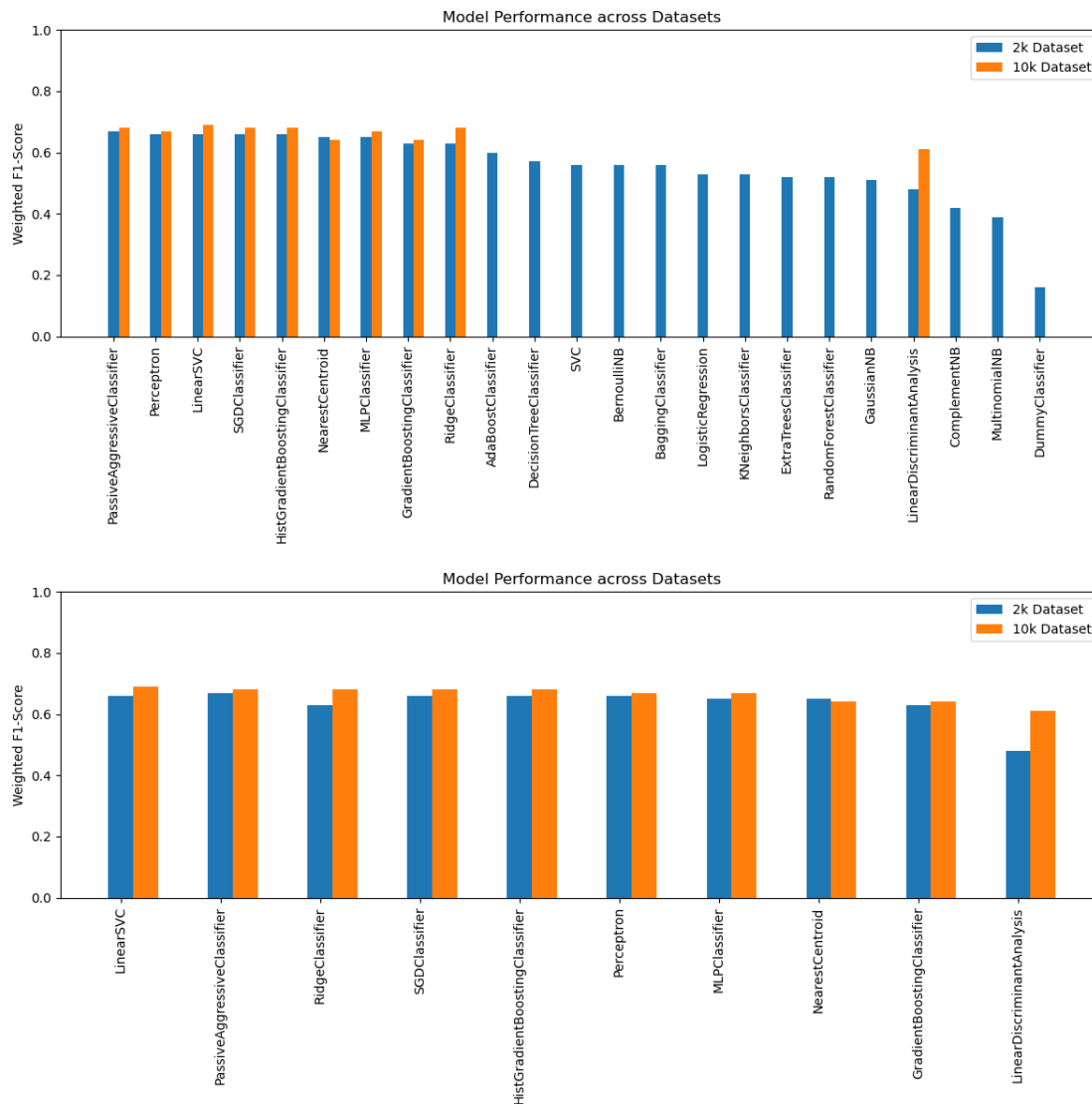
ingClassifier 6. NearestCentroid 7. MLPClassifier 8. GradientBoostingClassifier 9. RidgeClassifier 10. AdaBoostClassifier (because of an evaluation mistake, we used LinearDiscriminantAnalysis instead)

That gave us the following results:



The top 5 are the same, with the only exception of Perceptron falling behind against the RidgeClassifier. When comparing these models between datasets, it is evident that a bigger dataset yields better performance (for exponentially higher compute and time cost). Only NearestCentroid lost

performance when comparing the Datasets.



The final contenders are LinearSVC and PassiveAggressiveClassifier, which we would compare against each other using k-fold cross validation with different hyperparameters, but since training the model on the dataset takes a lot of time and a big strain on our computers, we will stop here and use the LinearSVC Classifier.

3.1 Model Selection

As a game can have multiple genres, our Model(s) has to be capable of multi-label-classification. sklearn's MultiOutputClassifier can do this. As a backend for MultiOutputClassifier we use LinearSVC

```
[13]: from sklearn.svm import LinearSVC
      from sklearn.multioutput import MultiOutputClassifier
```

```

multi_target_clf = MultiOutputClassifier(LinearSVC(max_iter=1337,
↳random_state=0), n_jobs=1)

multi_target_clf.fit(X_train, y_train)

y_pred = multi_target_clf.predict(X_test)

```

4 Evaluation

We evaluate our model by comparing the test data with the predicted data. We are using the worst case scenario by setting `zero_division=0.0` in the classification report. This means that if a metric cannot be calculated due to division by zero, it is set to 0.0. Setting this parameter to 1.0 (best case) does not significantly change the results.

Our approach involves training one model per genre, resulting in a total of 12 models. Each model predicts a specific genre, and the combined results of all models are shown at the bottom of the report. The input features are represented by `x`, and the output labels by `y`.

Key metrics such as precision and recall are calculated for each class. These metrics indicate whether all classes are recognized and how accurate the predictions are. Notably, only one class achieves perfect 1.0 precision. For some reason, the Early Access class performs particularly poorly. The F1 score is also included in the evaluation, as it provides a balanced measure of precision and recall. The support column indicates the number of samples for each class.

It is noteworthy that some of the top 10 words influencing the decision process are related to brands, such as “ea” in Sports, even though we removed the developer and publisher columns. Some words, like “brokkoli” in Racing, are not obviously related to the genre, which may indicate slight overfitting or the presence of only a few relevant but fitting data points in the dataset.

Generally, a model is considered very good with an F1 score above 0.8, and good with a score above 0.7. In our case, the F1 scores are 0.69 and 0.54, which means our model performs moderately well up to good. The low macro and micro scores are mainly due to problematic classes, but overall, the weighted average and samples average are quite acceptable.

```

[14]: from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred, target_names=y_test.columns,
↳zero_division=0.0))

feature_names = vectorizer.get_feature_names_out()
class_names = y_test.columns

for i, class_name in enumerate(class_names):
    coef = multi_target_clf.estimators_[i].coef_.flatten()
    # print the top 10 coefficients used
    top10 = np.argsort(coef)[-10:]
    print(f"Most important words of class '{class_name}':")
    print([feature_names[j] for j in top10[::-1]])
    print()

```

	precision	recall	f1-score	support
Action	0.86	0.87	0.87	300
Adventure	0.74	0.66	0.70	216
Casual	0.79	0.22	0.35	86
Early Access	0.50	0.02	0.04	46
Free To Play	0.79	0.28	0.41	83
Indie	0.77	0.81	0.79	245
Massively Multiplayer	0.89	0.19	0.31	42
RPG	0.80	0.55	0.65	127
Racing	1.00	0.58	0.74	12
Simulation	0.86	0.50	0.64	127
Sports	1.00	0.29	0.44	14
Strategy	0.80	0.41	0.54	106
micro avg	0.81	0.60	0.69	1404
macro avg	0.82	0.45	0.54	1404
weighted avg	0.80	0.60	0.65	1404
samples avg	0.81	0.66	0.69	1404

Most important words of class 'Action':

['action', 'weapons', 'shooter', 'fighting', 'fight', 'weapon', 'players', 'aim', 'gun', 'intense']

Most important words of class 'Adventure':

['adventure', 'explore', 'puzzles', 'smite', 'far', 'stories', 'remake', 'hunting', 'don', 'secrets']

Most important words of class 'Casual':

['puzzle', 'color', 'ball', 'smite', 'poker', 'click', 'communication', 'idle', 'cats', 'fun']

Most important words of class 'Early Access':

['early', 'pals', 'backrooms', 'automation', 'rotwood', 'access', 'design', 'vrchat', 'nephelym', 'idleon']

Most important words of class 'Free To Play':

['free', 'royale', 'mmo', 'pvp', 'arena', 'mmorpg', 'idle', 'cats', 'millions', 'team']

Most important words of class 'Indie':

['game', 'horror', 'building', 'different', 'vermintide', 'generated', 'roguelike', 'better', 'soundtrack', 'procedurally']

Most important words of class 'Massively Multiplayer':

['royale', 'mmorpg', 'players', 'mmo', 'pvp', 'ball', 'smite', 'scp', 'temtem', 'join']

Most important words of class 'RPG':

```
['rpg', 'loot', 'dungeons', 'combat', 'dungeon', 'character', 'fantasy',  
'quests', 'skills', ' ']
```

Most important words of class 'Racing':

```
['cars', 'racing', 'car', 'race', 'speed', 'driving', 'brokkoli', 'ddnet',  
'rally', 'jeff']
```

Most important words of class 'Simulation':

```
['simulator', 'realistic', 'simulation', 'physics', 'sandbox', 'building',  
'workshop', 'management', 'car', 'idle']
```

Most important words of class 'Sports':

```
['racing', 'skate', 'sports', 'football', 'rally', 'virtual', 'ea', 'vrchat',  
'hunting', 'realistic']
```

Most important words of class 'Strategy':

```
['strategy', 'turn', 'units', 'buildings', 'strategic', 'heroes', 'tactical',  
'command', ' ', 'squad']
```

5 Optimization

- Since our dataset contains multiple languages, it would be beneficial to either train a separate model for each language or to standardize the data before and remove the stop words specific to each language.
- Hyperparameter validation should also be performed. For example, in LinearSVC, the C parameter controls the learning rate and could be further optimized.
- Instead of a simple train-test split, k-fold cross validation should be used to achieve better data mixing and more robust results.
- Additionally, ensemble learning methods could be considered to further improve performance.

The biggest limitation of our dataset is the presence of too many languages but too few entries for each, which is also constrained by our computing resources.

6 Conclusion and outlook

To conclude we can say that our model performs reasonably well for the intended application. With a larger dataset, the results would likely improve further. Considering the points mentioned above, it is quite impressive that the model achieves these results using only a small dataset and limited computational resources.

Our collaboration as a team worked very smoothly throughout the project. Communication and planning were effective, allowing us to coordinate our tasks efficiently and make steady progress.

The main challenge we faced was the limited computational resources available to us. Especially when working with the 10k dataset, training the models for statistical evaluation took a considerable

amount of time. To address this, each team member ran different models in parallel on their own machines, with some training processes running for several days.

Due to these computational constraints, we decided not to process the full dataset with 80,000 entries. Even though we had access to very powerful PCs equipped with the latest high-end components, the training times were still prohibitively long. As a result, we focused our efforts on the smaller datasets to ensure we could complete the project within a reasonable timeframe.

In summary, this project provided us with valuable insights into the challenges and opportunities of machine learning in a real-world context. Despite the limitations we faced, we were able to develop a functioning model and gain practical experience in data preprocessing, model selection, and evaluation. We are proud of what we achieved as a team and look forward to applying the knowledge and skills gained here to future projects.