

Classification Progress

Machine Learning Project

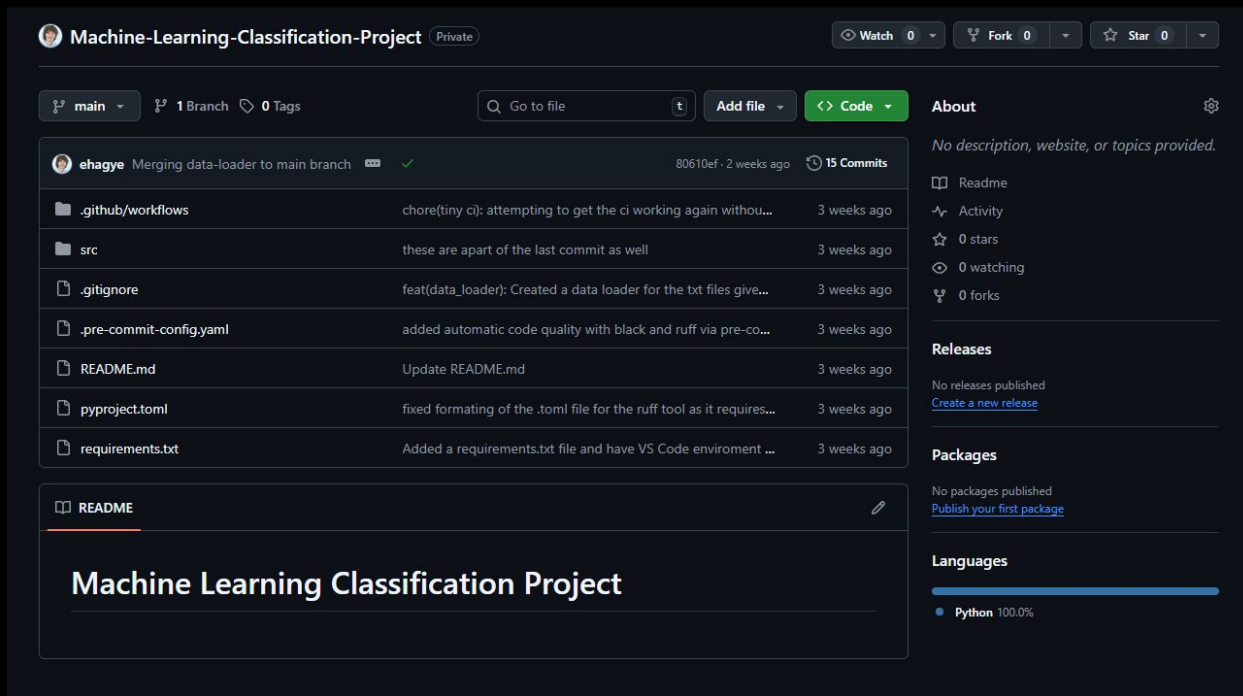
November 2025

Team Members:
Naol Seyum
Emery Hagye

Github Creation with Added Features



- Created a simple Github Repository for keeping progress
- Used Git Actions to create a Tiny CL
- Implemented a pre-commit to help further with formatting



The screenshot shows a GitHub repository page for 'Machine-Learning-Classification-Project' (Private). The repository has 1 branch (main), 0 tags, 0 forks, and 0 stars. The repository is owned by 'ehagye' and has 15 commits. The repository is currently merging 'data-loader' to the 'main' branch. The repository contains the following files and folders:

File/Folder	Description	Commit Date
.github/workflows	chore(tiny ci): attempting to get the ci working again without...	3 weeks ago
src	these are apart of the last commit as well	3 weeks ago
.gitignore	feat(data_loader): Created a data loader for the txt files give...	3 weeks ago
.pre-commit-config.yaml	added automatic code quality with black and ruff via pre-co...	3 weeks ago
README.md	Update README.md	3 weeks ago
pyproject.toml	fixed formatting of the .toml file for the ruff tool as it requires...	3 weeks ago
requirements.txt	Added a requirements.txt file and have VS Code enviroment ...	3 weeks ago

The repository has a README file. The README content is:

Machine Learning Classification Project

The repository also has a sidebar with the following sections:

- About**: No description, website, or topics provided.
- Readme**: Readme
- Activity**: Activity
- Stars**: 0 stars
- Watching**: 0 watching
- Forks**: 0 forks
- Releases**: No releases published. [Create a new release](#)
- Packages**: No packages published. [Publish your first package](#)
- Languages**: Python 100.0%

Data Loader Python File

- Created a Data Loader that Automatically finds correct path
- Can easily be reused anywhere in the project
- A little preprocessing with replacing the missing entries with nan

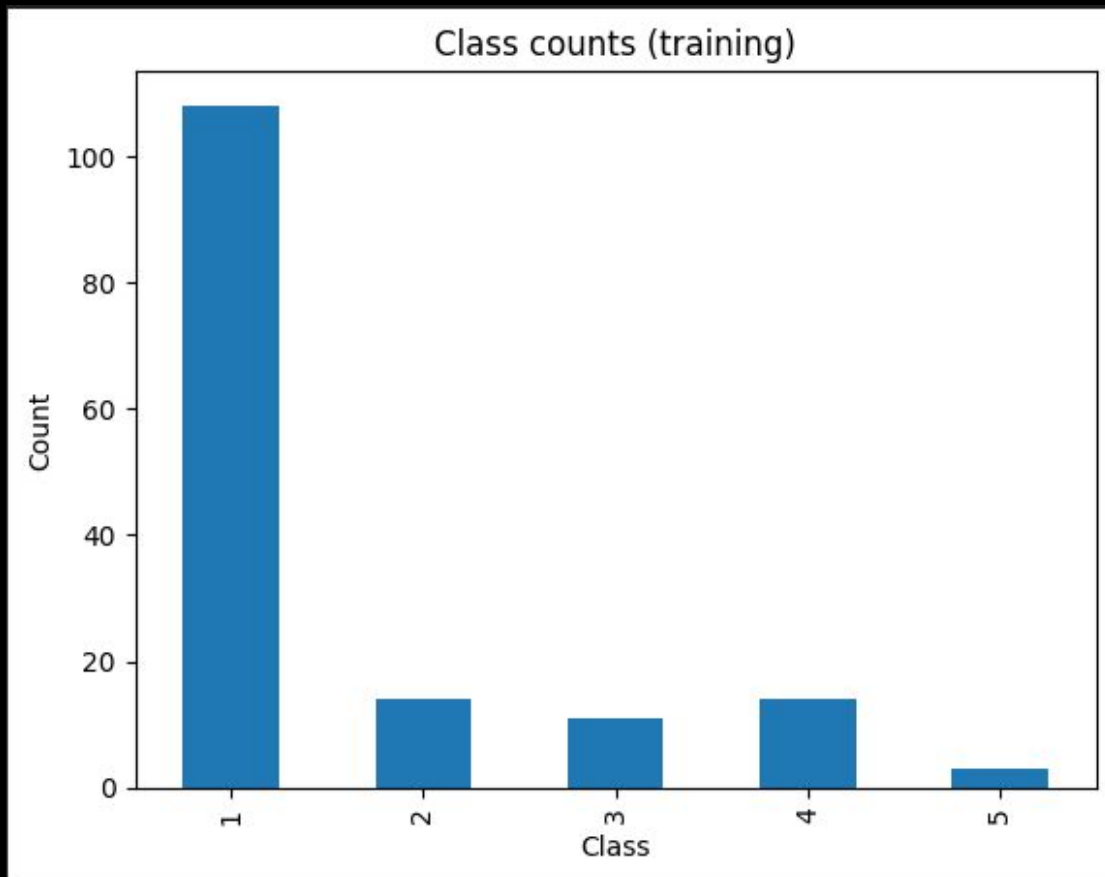


```
src > data_loader.py > load_dataset
1  from pathlib import Path
2
3  import numpy as np
4
5  missing_entry = 1e99
6
7  def _project_root() -> Path:
8      return Path(__file__).resolve().parents[1]
9
10 def _resolve(pathlike) -> Path:
11     p = Path(pathlike)
12     if not p.is_absolute():
13         p = _project_root() / p
14     return p
15
16 def load_dataset(train_path, label_path=None, test_path=None):
17     """
18     Loads training/test datasets and changes the missing values (1e99) to be Nan.
19
20     Parameters
21     -----
22     train_path: a string
23         this is a path to the training data file.
24     label_path: string or none
25         this is a path to the training labels file.
26     test_path: string or none
27         this is a path to the test data file.
28
29     Returns
30     -----
31     dict type
32         A dictionary with the keys X_train, Y_train, and X_test.
33         (only when they are given as parameters)
34     """
35     # loading the data from the given files and replacing the missing entries with NaN
36     def load(path):
37         path = _resolve(path)
38
39         arr = np.loadtxt(path)
40         arr = np.where(arr == missing_entry, np.nan, arr)
41         return arr
42
43     data = {"X_train": load(train_path)}
44     if label_path:
45         data["y_train"] = load(label_path).astype(int).ravel()
46     if test_path:
47         data["X_test"] = load(test_path)
48     return data["X_train"], data.get("y_train"), data.get("X_test")
```

Began EDA of Dataset 1



```
Total Nans: 9,936
Top 10 features by most Nans:
f1    12
f1    10
f1     9
f1     9
f1     9
f1     9
f1     9
f1     9
f1     9
f1     8
dtype: int64
Top 10 rows with most missing values:
58    85
51    85
89    83
109   82
140   81
47    80
73    79
133   79
117   78
71    78
dtype: int64
```



Methods to use in the future



- For Datasets 1 and 2 I will use aggressive Dimensionality reduction for the large number of features
 - Variance thresholding
 - PCA
- Complex Model Selection
 - Linear SVM
 - Random Forest
 - Neural Network



Spam Email Detection

Machine Learning Project

November 2025

Team Members:
Naol Seyum
Emery Hagye

Why Spam Detection?

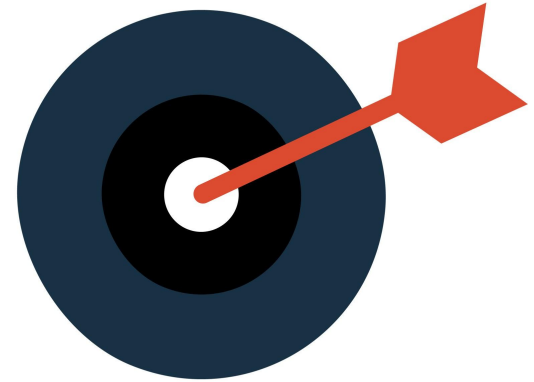
- Inboxes overflow with spam and scams.



- Spam wastes time and risks security.



- ML can detect spam automatically.



- Goal: Build a model to keep inboxes clean.

Project Workflow



Emails

- Combined two training datasets.



Preprocessing

- Cleaned and vectorized text.



Feature Extraction

- Trained an SVM model.



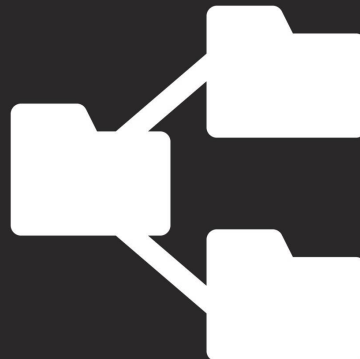
Feature Extraction

- Tuned with cross-validation.



Predictions

- Predicted spam or ham on test data.



Data Preprocessing

```
9 import pandas as pd
10
11 # step-1 data loading part
12 # this part just reads the files and makes s
13 train1 = pd.read_csv("spam_train1.csv") # columns v1 (label), v2 (text)
14 train2 = pd.read_csv("spam_train2.csv") # columns label, text, label_num, etc.
15 test = pd.read_csv("spam_test.csv") # column message
16
17 # step-2 cleaning up the data
18 # just renaming and merging both train sets so everything looks the same.
19 train1_simple = train1[['v1', 'v2']].rename(columns={'v1': 'label', 'v2': 'text'})
20 train2_simple = train2[['label', 'text']]
21 data = pd.concat([train1_simple, train2_simple], ignore_index=True)
22
23 # step-3 turning labels into numbers.
24 # ham = 0, spam = 1 so the model can understand it.
25 data['label_num'] = data['label'].map({'ham': 0, 'spam': 1})
26
```

Merging two training datasets.

- Merged two training datasets into one.

Encoded labels

- ham = 0
- spam = 1

Hashing Vectorizer

```
27 # step-4 feature setup
28 # hashing vectorizer turns words into numbers, linear SVM does the classification.
29 from sklearn.feature_extraction.text import HashingVectorizer
30 from sklearn.svm import LinearSVC
31 from sklearn.pipeline import make_pipeline
32
33 pipe = make_pipeline(
34     HashingVectorizer(
35         n_features=2**18,
36         alternate_sign=False,
37         ngram_range=(1, 2) # unigrams + bigrams
38     ),
39     LinearSVC()
40 )
```

- Converted text into numeric features

Renamed columns

- v1 → label
- v2 → text

Training the Model & Checking Performance

- Split data = 80% for training and 20% for validation.
- Trained Support Vector Machine (SVM).
- Checked Accuracy, Precision, Recall, and F1-score.
- Used GridSearchCV for tuning (C = [0.1, 1, 10]).

True Positive (TP) → spam predicted as spam.
False Positive (FP) → ham predicted as spam.
False Negative (FN) → spam predicted as ham.

```
42 # step-5 training and validation split.
43 # splitting the data so I can test the model's performance before final training.
44 from sklearn.model_selection import train_test_split
45
46 X_train, X_val, y_train, y_val = train_test_split(
47     data['text'], data['label_num'],
48     test_size=0.2,
49     stratify=data['label_num'],
50     random_state=42
51 )
52 # step-6 training the model and checking accuracy.
53 # fitting the model and then printing accuracy, precision, recall, and f1.
54 from sklearn.metrics import accuracy_score, precision_recall_fscore_support, roc_auc_score
55
56 pipe.fit(X_train, y_train)
57 y_pred = pipe.predict(X_val)
58 acc = accuracy_score(y_val, y_pred)
59 prec, rec, f1, _ = precision_recall_fscore_support(y_val, y_pred, average='binary')
60
61 print("Baseline LinearSVC + HashingVectorizer")
62 print(f"Accuracy: {acc:.4f}")
63 print(f"Precision: {prec:.4f}")
64 print(f"Recall: {rec:.4f}")
65 print(f"F1-score: {f1:.4f}")
66 # step-7 tuning time.
67 # trying different C values to see which one gives the best spam detection score
68 from sklearn.model_selection import GridSearchCV
69
70 param_grid = [
71     {'linearsvc__C': [0.1, 1, 10]}
72 ]
73 grid = GridSearchCV(
74     pipe,
75     param_grid=param_grid,
76     cv=5,
77     scoring='f1' # bias towards good spam detection.
78 )
79
80 grid.fit(data['text'], data['label_num'])
81 best_model = grid.best_estimator_
82
83 print("Best C from GridSearchCV:", grid.best_params_)
```

```

85 # step-8 final training and test prediction
86 # model done training, now predicting on the test file and saving it to csv
87 best_model.fit(data['text'], data['label_num'])
88
89 test_text = test['message']
90 test_preds = best_model.predict(test_text)
91
92 # step-9 mapping numbers back to labels
93 # just converting 0 and 1 back to ham and spam so it's readable
94 label_map = {0: 'ham', 1: 'spam'}
95 pred_labels = [label_map[p] for p in test_preds]
96
97 # step-10 saving the result
98 # final output with id and predicted label
99 output = pd.DataFrame({
100     'id': range(len(test)),
101     'label': pred_labels
102 })
103
104 output.to_csv("spam_test_predictions.csv", index=False)
105 print("The prediction result is saved to spam_test_predictions.csv")
106
107 # step-11 quick summary of prediction counts
108 # just counting how many are spam vs ham in the final predictions
109 ham_count = (output['label'] == 'ham').sum()
110 spam_count = (output['label'] == 'spam').sum()
111
112 print(f"\nSummary of predictions:")
113 print(f"Ham emails : {ham_count}")
114 print(f"Spam emails: {spam_count}")

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
naolseyum@Mac Spam Email Detection 2 % python3 ham_spam_email_detection.py
Baseline LinearSVC + HashingVectorizer
Accuracy: 0.9535
Precision: 0.9620
Recall: 0.8172
F1-score: 0.8837
Best C from GridSearchCV: {'linearsvc__C': 10}
The prediction result is saved to spam_test_predictions.csv
Summary of predictions:
Ham emails : 5248
Spam emails: 1199

```

Model accuracy

Precision

Best parameter

Ham: X

Spam: Y

• 95.3%

• 96.2%

• C=10

• 5248

• 1199

• Predictions saved to: spam_test_predictions.csv

Model Performance & Output Summary