Classification Progress

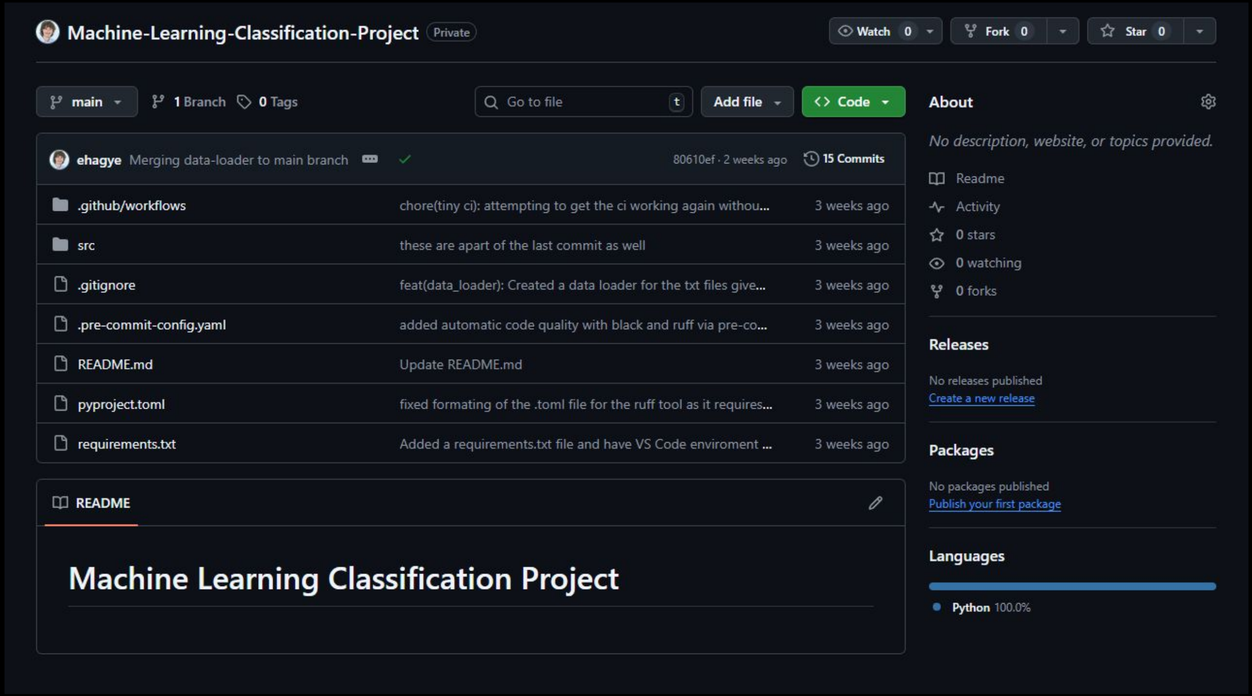
Machine Learning Project

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



Data Loader Python File

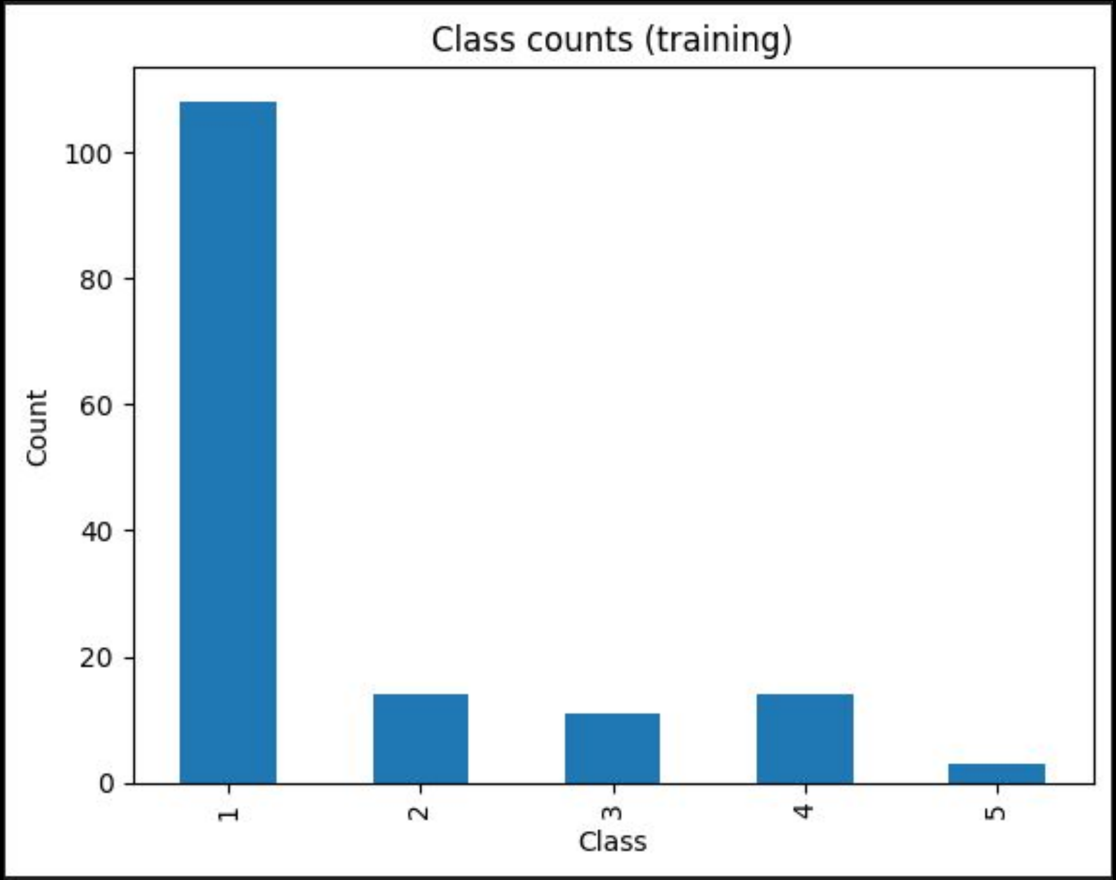
- Created a Data Loader that Automatically finds correct path
- Can easily be reused anywhere in the project
- A little preprocessing with replacing the missing entries with nan

src > data_loader.py > load_dataset

```python
from pathlib import Path

import numpy as np

missing_entry = 1e99

def _project_root() -> Path:
    return Path(__file__).resolve().parents[1]

def _resolve(pathlike) -> Path:
    p = Path(pathlike)
    if not p.is_absolute():
        p = _project_root() / p
    return p

def load_dataset(train_path, label_path=None, test_path=None):
    """
    Loads training/test datasets and changes the missing values (1e99) to be Nan.

    Parameters
    -----------
    train_path: a string
        this is a path to the training data file.
    label_path: string or none
        this is a path to the training labels file.
    test_path: string or none
        this is a path to the test data file.

    Returns
    --------
    dict type
        A dictionary with the keys X_train, Y_train, and X_test.
        (only when they are given as parameters)
    """
    # loading the data from the given files and replacing the missing entries with NaN
    def load(path):
        path = _resolve(path)

        arr = np.loadtxt(path)
        arr = np.where(arr == missing_entry, np.nan, arr)
        return arr

    data = {"X_train": load(train_path)}
    if label_path:
        data["y_train"] = load(label_path).astype(int).ravel()
    if test_path:
        data["X_test"] = load(test_path)
    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

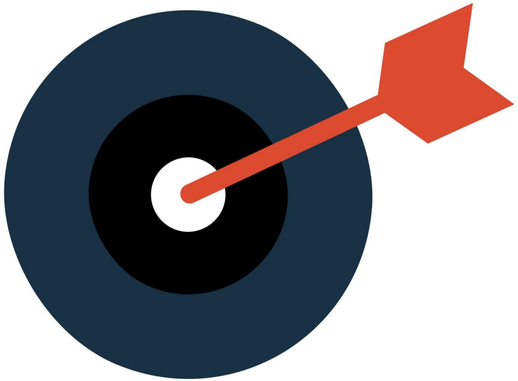
Team Members:
Naol Seyum
Emery Hagye

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

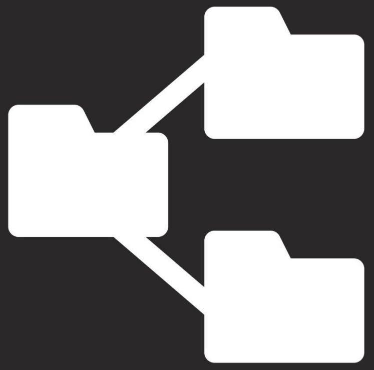
- **Trained an SVM model.**

🔢 Feature Extraction

- **Tuned with cross-validation.**

✅ Predictions

- **Predicted spam or ham on test data.**



Data Preprocessing

```python
import pandas as pd

# step-1 data loading part
# this part just reads the files and makes s
train1 = pd.read_csv("spam_train1.csv")    # columns v1 (label), v2 (text)
train2 = pd.read_csv("spam_train2.csv")   # columns label, text, label_num, etc.
test   = pd.read_csv("spam_test.csv")   # column message

# step-2 cleaning up the data
# just renaming and merging both train sets so everything looks the same.
train1_simple = train1[['v1', 'v2']].rename(columns={'v1': 'label', 'v2': 'text'})
train2_simple = train2[['label', 'text']]
data = pd.concat([train1_simple, train2_simple], ignore_index=True)

# step-3 turning labels into numbers.
# ham = 0, spam = 1 so the model can understand it.
data['label_num'] = data['label'].map({'ham': 0, 'spam': 1})
```

Merging two training datasets.

- Merged two training datasets into one.

Encoded labels

- ham = 0
- spam = 1

```python
# step-4 feature setup
  # hashing vectorizer turns words into numbers, linear SVM does the classification.
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline

pipe = make_pipeline(
    HashingVectorizer(
        n_features=2**18,
        alternate_sign=False,
        ngram_range=(1, 2)  # unigrams + bigrams
    ),
    LinearSVC()
)
```

Hashing Vectorizer

- 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.

```python
# step-5 training and validation split.
  # splitting the data so I can test the model’s performance before final training.
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    data['text'], data['label_num'],
    test_size=0.2,
    stratify=data['label_num'],
    random_state=42
)
# step-6 training the model and checking accuracy.
  # fitting the model and then printing accuracy, precision, recall, and f1.
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, roc_auc_score

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_val)
acc = accuracy_score(y_val, y_pred)
prec, rec, f1, _ = precision_recall_fscore_support(y_val, y_pred, average='binary')

print("Baseline LinearSVC + HashingVectorizer")
print(f"Accuracy:  {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall:    {rec:.4f}")
print(f"F1-score:  {f1:.4f}")
# step-7 tuning time.
  # trying different C values to see which one gives the best spam detection score
from sklearn.model_selection import GridSearchCV

param_grid = {
    'linearsvc__C': [0.1, 1, 10]
}
grid = GridSearchCV(
    pipe,
    param_grid=param_grid,
    cv=5,
    scoring='f1'  # bias towards good spam detection.
)

grid.fit(data['text'], data['label_num'])
best_model = grid.best_estimator_

print("Best C from GridSearchCV:", grid.best_params_)
```

```python
    # step-8 final training and test prediction
# model done training, now predicting on the test file and saving it to csv
best_model.fit(data['text'], data['label_num'])

test_text = test['message']
test_preds = best_model.predict(test_text)

  # step-9 mapping numbers back to labels
# just converting 0 and 1 back to ham and spam so it's readable
label_map = {0: 'ham', 1: 'spam'}
pred_labels = [label_map[p] for p in test_preds]

  # step-10 saving the result
# final output with id and predicted label
output = pd.DataFrame({
    'id': range(len(test)),
    'label': pred_labels
})

output.to_csv("spam_test_predictions.csv", index=False)
print("The prediction result is saved to spam_test_predictions.csv")

# step-11 quick summary of prediction counts
# just counting how many are spam vs ham in the final predictions
ham_count = (output['label'] == 'ham').sum()
spam_count = (output['label'] == 'spam').sum()

print(f"\nSummary of predictions:")
print(f"Ham emails : {ham_count}")
print(f"Spam emails: {spam_count}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS zsh

```
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 Performance & Output Summary

Model accuracy	Precision	Best parameter	Ham: X	Spam: Y
• 95.3%	• 96.2%	• C=10	• 5248	• 1199

- Predictions saved to: `spam_test_predictions.csv`