gender_classification

January 20, 2025

1 Reddit user gender classification

1.0.1 Libraries and configuration

[1]:

```python
from time import time

import joblib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.sparse as sp
from sklearn.base import BaseEstimator
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import HalvingGridSearchCV, cross_val_score
from sklearn.model_selection import train_test_split
```

[2]:

```python
plt.style.use(['grid', 'science', 'notebook', 'mylegend'])

data_dir = 'data'
```

1.1 Load the training and test data

[3]:

```python
def load_data(load_test: bool) -> tuple[pd.DataFrame, pd.DataFrame, pd.
  ↪DataFrame]:
    train_data = pd.read_csv(f'{data_dir}/train_data.csv')
    target = pd.read_csv(f'{data_dir}/train_target.csv')
    if load_test:
        test_data = pd.read_csv(f'{data_dir}/test_data.csv')
    else:
        test_data = pd.DataFrame()
    return train_data, target, test_data
```

[4]:

```python
train_data, target, test_data = load_data(load_test=True)

print(f"Number of authors in training set: {train_data["author"].unique().
  ↪shape[0]}")
```

```
Number of authors in training set: 5000
```

1.2 Feature extraction

[5]:

```
def create_subreddit_idx(data: pd.DataFrame) -> pd.Series:
    """Map every subreddit to a unique integer."""
    subreddits = data["subreddit"].unique()
    return pd.Series(index=subreddits, data=np.arange(len(subreddits)))
```

[6]:

```
def extract_subreddits(
    author_data: pd.DataFrame,
    subreddit_idx: pd.Series,
) -> sp.csr_array:
    """
    This function converts all the subreddits the author has posted in into a␣
 ↪sparse
    array of length N (where N is the number of subreddits in the dataset) with␣
 ↪1s in
    the indexes of the subreddits the author has posted in.
    """
    user_subs = author_data["subreddit"]
    subs_in_idx = user_subs.isin(subreddit_idx.index)
    user_subs = user_subs[subs_in_idx].to_numpy()

    # idxs is an array with the indexes of the subreddits in subreddits_idx
    idxs = subreddit_idx.loc[user_subs].to_numpy()

    # create a sparse array indicating the subreddits the author has posted in
    v = sp.dok_array((1, len(subreddit_idx)))  # dok = dictionary of keys
    for idx in idxs:
        v[0, idx] = 1
    return v.tocsr()  # convert to compressed sparse row format
```

[7]:

```
def extract_text(author_data: pd.DataFrame) -> str:
    """Returns all the posts of an author as a single string."""
    group_text = author_data["body"].astype(str).to_numpy()
    return " ".join(group_text)
```

[8]:

```
def vectorize_text(
    vectorizer: TfidfVectorizer,
    text: list[str],
    data_is_test: bool,
) -> sp.csr_array:
    """
    This function vectorizes the text of an author using the provided␣
 ↪vectorizer.
    If the data is test data, the vectorizer is only transformed, otherwise it␣
 ↪is fit
    and transformed.
    """
    if data_is_test:
        return vectorizer.transform(text)
    else:
        return vectorizer.fit_transform(text)
```

[9]:

```
def extract_features(
    data: pd.DataFrame,
    subreddit_idx: pd.Series,
    vectorizer: TfidfVectorizer,
    *,
    target: pd.DataFrame | None = None,
) -> tuple[sp.csr_matrix, pd.Series] | sp.csr_matrix:
    """Extract features from the data."""

    data_is_test = True if target is None else False

    subs_dict: dict[str, sp.csr_array] = {}
    for author, group in data.groupby("author"):
        subs_dict[author] = extract_subreddits(group, subreddit_idx)

    if data_is_test:
        authors = data["author"].unique()
    else:
        authors = target["author"]

    # Generate a sparse matrix with the authors as rows
    # and the subreddits they have posted in as columns
    subs_matrix: sp.csr_matrix = sp.vstack([subs_dict[author] for author in␣
 ↪authors])

    text_dict: dict[str, str] = {}
    for author, group in data.groupby("author"):
        text_dict[author] = extract_text(group)

    author_text: list[str] = [text_dict[author] for author in authors]
    text_features = vectorize_text(vectorizer, author_text, data_is_test)

    # print(type(text_features))

    X = sp.hstack([subs_matrix, text_features])

    if data_is_test:
        return X
    else:
        y: pd.Series = target["gender"]
        return X, y
```

[10]:
```
subreddit_idx = create_subreddit_idx(train_data)
vectorizer = TfidfVectorizer(max_df=0.95, stop_words="english",␣
  ↪max_features=10000)
```

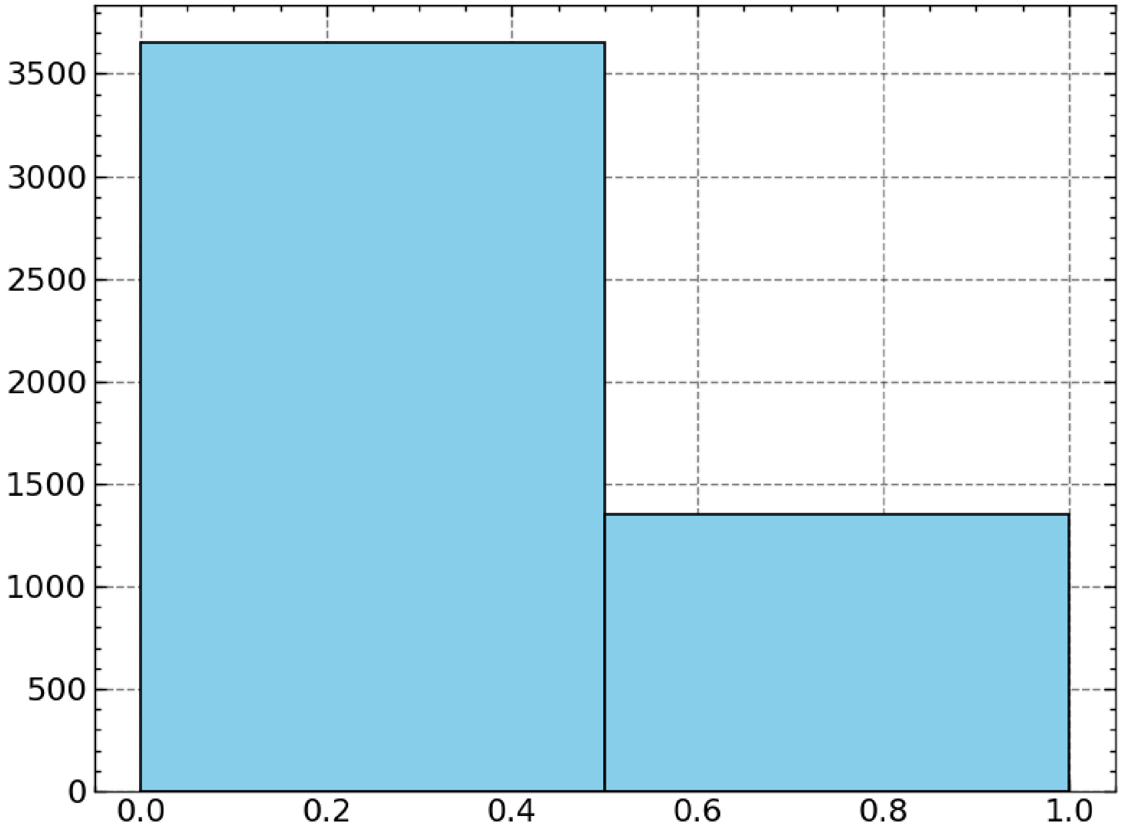
[11]:
```
X, y = extract_features(train_data, subreddit_idx, vectorizer, target=target)
```

[12]:
```
X = X.toarray()  # needed for Naive Bayes
```

1.2.1 Dataset exploration

[13]:
```
fig, ax = plt.subplots(1, 1)
ax.hist(
    target["gender"],
    bins=2,
    color="skyblue",
    edgecolor="black",
    linewidth=1.2,
    align="mid",
)

fig.tight_layout()
plt.show()
```



The dataset is strongly unbalanced towards the male class.

1.3 Model selection

Define a set of models to try on the dataset. Then, for each model, perform hyperparameters tuning using `GridSearchCV`. Finally, pick the best model overall.

[14]:

```python
from sklearn.linear_model import LogisticRegression as LogReg
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.naive_bayes import (
    GaussianNB as NB,
    MultinomialNB as MNB,
    ComplementNB as CNB,
)
from sklearn.tree import DecisionTreeClassifier as DT
from sklearn.ensemble import GradientBoostingClassifier as GBC
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.neural_network import MLPClassifier as MLP

classifiers: dict[str, BaseEstimator] = {
    "LogReg": LogReg(n_jobs=-1, random_state=42),
    "KNN": KNN(n_jobs=-1),
    "Naive Bayes": NB(),
    "Multinomial NB": MNB(),
    "Complement NB": CNB(),
    "Random Forest": RF(n_jobs=-1, random_state=42),
    "Decision Tree": DT(random_state=42),
    "Gradient Boosting": GBC(random_state=42),
    "MultiLayer Perceptron": MLP(solver="adam", random_state=42),
    # "SVM": SVC(probability=True),  # takes too long to train and doesn't
                                     # yield the best results
}
```

Get a benchmark of the performances of the various models with default parameters on a subset of the training set. The slower/less accurate models will be discarded to improve training time.

[15]:

```python
X_bench_train, X_bench_val, y_bench_train, y_bench_val = train_test_split(
    X[::3], y[::3], test_size=0.2, random_state=42
)
```

[16]:

```python
longest_name: int = max(len(key) for key in classifiers.keys())
benchmark_times: dict[str, float] = {}
benchmark_scores: dict[str, float] = {}
for name, clf in classifiers.items():
    t_start = time()
    clf.fit(X, y)
    t_end = time()
    bench_time = t_end - t_start
    score = clf.score(X_bench_val, y_bench_val)
    benchmark_times[name] = bench_time
    benchmark_scores[name] = score
    print(f"Benchmarking {name:{longest_name}} -- time: {bench_time:6.2f}s,␣
 ↪score = {score:.3f}")
```

```
Benchmarking LogReg               -- time:   8.82s, score = 0.934
Benchmarking KNN                  -- time:   0.04s, score = 0.832
Benchmarking Naive Bayes          -- time:   1.13s, score = 0.829
Benchmarking Multinomial NB       -- time:   0.09s, score = 0.916
Benchmarking Complement NB        -- time:   0.10s, score = 0.913
Benchmarking Random Forest        -- time:   7.20s, score = 1.000
Benchmarking Decision Tree        -- time:  29.76s, score = 1.000
Benchmarking Gradient Boosting    -- time: 344.22s, score = 0.865
Benchmarking MultiLayer Perceptron -- time:  54.97s, score = 1.000
```

Remove the two classifiers with the lowest score and the two classifiers with the highest training time.

[17]:

```
lowest_scoring_models: list[str] = sorted(
    benchmark_scores, key=benchmark_scores.get
)[:2]
slowest_models: list[str] = sorted(
    benchmark_times, key=benchmark_times.get, reverse=True
)[:2]
models_to_remove = set(lowest_scoring_models + slowest_models)

for model in models_to_remove:
    classifiers.pop(model, None)

print("Classifiers to perform GridSearch on:")
for name in classifiers.keys():
    print(f"- {name}")
print("Removed models:")
for name in models_to_remove:
    print(f"- {name}")
```

```
Classifiers to perform GridSearch on:
- LogReg
- Multinomial NB
- Complement NB
- Random Forest
- Decision Tree
Removed models:
- KNN
- MultiLayer Perceptron
- Gradient Boosting
- Naive Bayes
```

[18]:

```
param_grids: list[dict[str, np.ndarray]] = [
    {"C": np.logspace(-3, 2)},  # LogReg, C = 1/lambda, lambda = regularization
    {"alpha": np.logspace(-2, 2)},  # Multinomial Naive Bayes, alpha = smoothing
    {"alpha": np.logspace(-2, 2)},  # Complement Naive Bayes, alpha = smoothing
    {"n_estimators": np.arange(1, 100, 5)},  # Random Forest
    {
        "max_depth": np.arange(1, 10),
        "min_samples_split": np.arange(2, 5),
    },  # Decision Tree
]
```

[19]:

```
longest_name: int = max(len(key) for key in classifiers.keys())
best_clfs: dict[str, BaseEstimator] = {}
best_pars: dict[str, dict[str, float]] = {}
for (name, clf), param_grid in zip(classifiers.items(), param_grids):
    print(f"Training {name:{longest_name}} -- ", end="")
    search = HalvingGridSearchCV(
        clf,
        param_grid,
        cv=5,
        scoring="roc_auc",
        n_jobs=-1,
        random_state=42,
    )
    t_start = time()
    search.fit(X, y)
    t_end = time()
    best_clfs[name] = search.best_estimator_
    best_pars[name] = search.best_params_
    print(f"score = {search.best_score_:.3f} ({t_end - t_start:6.2f}s)")
```

```
Training LogReg         -- score = 0.910 ( 46.72s)
Training Multinomial NB -- score = 0.915 ( 11.08s)
Training Complement NB  -- score = 0.915 ( 11.06s)
Training Random Forest  -- score = 0.838 (118.38s)
Training Decision Tree  -- score = 0.653 ( 17.41s)
```

[20]:

```
best_scores_cv: dict[str, np.ndarray] = {}
for name, clf in best_clfs.items():
    print(f"Scoring {name:{longest_name}}", end=" ")
    t_start = time()
    scores = cross_val_score(clf, X, y, cv=5, scoring='roc_auc', n_jobs=-1)
    t_end = time()
    best_scores_cv[name] = scores
    print(f"({t_end - t_start:5.2f}s)")
```

```
Scoring LogReg         ( 7.69s)
Scoring Multinomial NB ( 1.34s)
Scoring Complement NB  ( 1.58s)
Scoring Random Forest  (21.33s)
Scoring Decision Tree  ( 4.66s)
```

[21]:

```
print("Classifier  " + " " * (longest_name - 10) + "Score")
for name, scores in best_scores_cv.items():
    print(f"{name:{longest_name}}  {scores.mean():.3f} +/- {scores.std():.3f}")
```

```
Classifier      Score
LogReg          0.910 +/- 0.016
Multinomial NB  0.915 +/- 0.014
Complement NB   0.915 +/- 0.014
Random Forest   0.838 +/- 0.013
Decision Tree   0.655 +/- 0.031
```

Now pick the best `num_ensemble` models, fine tune their parameters and use them to create an ensemble using `VotingClassifier`.

[22]:

```
num_ensemble = 3

sorted_models: list[tuple[str, np.ndarray]] = sorted(
    best_scores_cv.items(), key=lambda item: item[1].mean(), reverse=True
)
top_models: list[tuple[str, np.ndarray]] = sorted_models[:num_ensemble]
top_scores: dict[str, tuple[np.float64, np.float64]] = {
    name: (scores.mean(), scores.std()) for name, scores in best_scores_cv.
  ↪items()
}
top_models: dict[str, BaseEstimator] = {
    name: classifiers[name] for name, _ in top_models
}
longest_name: int = max(len(key) for key in top_models.keys())

print(f"The top {num_ensemble} models are:")
for name, _ in top_models.items():
    print(f"- {name:{longest_name}} -- score = {top_scores[name][0]:.3f},␣
  ↪params = {best_pars[name]}")
```

```
The top 3 models are:
- Complement NB  -- score = 0.915, params = {'alpha':
np.float64(0.1151395399326447)}
- Multinomial NB -- score = 0.915, params = {'alpha':
np.float64(0.1151395399326447)}
- LogReg         -- score = 0.910, params = {'C':
np.float64(0.9102981779915218)}
```

[23]:

```
fine_param_grids: list[dict[str, np.ndarray]] = [
    {"alpha": np.linspace(5e-2, 5e-1)},  # Complement Naive Bayes
    {"alpha": np.linspace(5e-2, 5e-1)},  # Multinomial Naive Bayes
    {"C": np.linspace(5e-1, 5e0)},  # LogReg
]
```

[24]:

```
top_pars: dict[str, dict[str, np.float64]] = {}
for (name, clf), param_grid in zip(top_models.items(), fine_param_grids):
    print(f"Training {name:{longest_name}} -- ", end="")
    search = HalvingGridSearchCV(
        clf,
        param_grid,
        cv=5,
        scoring="roc_auc",
        n_jobs=-1,
        random_state=42,
    )
    t_start = time()
    search.fit(X, y)
    t_end = time()
    top_models[name] = search.best_estimator_
    top_pars[name] = search.best_params_
    print(
        f"score = {search.best_score_:.3f} ({t_end - t_start:5.2f}s), pars =␣
 ↪{top_pars[name]}"
    )
```

```
Training Complement NB  -- score = 0.915 (10.27s), pars = {'alpha':
np.float64(0.1142857142857143)}
Training Multinomial NB -- score = 0.915 (11.88s), pars = {'alpha':
np.float64(0.1142857142857143)}
Training LogReg         -- score = 0.910 (47.33s), pars = {'C':
np.float64(1.1428571428571428)}
```

[25]:

```
from sklearn.ensemble import VotingClassifier

print("Creating ensemble with following models:")
for name in top_models.keys():
    print(f"- {name}")

ensemble = VotingClassifier(estimators=list(top_models.items()), voting="soft")
print(f"\nFitting ensemble ", end="")
t_start = time()
ensemble.fit(X, y)
t_end = time()
print(f"({t_end - t_start:.2f}s)")
t_start = time()
scores = cross_val_score(ensemble, X, y, cv=5, scoring="roc_auc")
t_end = time()
```

```
print(
    f"Ensemble method score = {scores.mean():.3f} +/- {scores.std():.3f}␣
 ↪({t_end - t_start:.2f}s)"
)
```

```
Creating ensemble with following models:
- Complement NB
- Multinomial NB
- LogReg

Fitting ensemble (4.50s)
Ensemble method score = 0.922 +/- 0.014 (21.37s)
```

[26]:

```
joblib.dump(ensemble, f"{data_dir}/ensemble_clf.joblib")
```

[26]:

```
['data/ensemble_clf.joblib']
```

1.4 Preparing the solution

[27]:

```
X_test = extract_features(test_data, subreddit_idx, vectorizer)
```

[28]:

```
y_pred = ensemble.predict_proba(X_test)[:, 1]
```

[29]:

```
solution = pd.DataFrame({"author": test_data.author.unique(), "gender": y_pred})
solution.head()
```

[29]:

```
          author    gender
0  ejchristian86  0.999991
1      ZenDragon  0.001040
2   savoytruffle  0.003738
3   hentercenter  0.021227
4   rick-o-suave  0.094984
```

[30]:

```
solution.to_csv("submission_ensemble.csv", index=False)
```

Now go to Kaggle, click “Submit Prediction” and upload the file “submission.csv” to see the test score.