Hate Speech Detection - scikit-learn

Ravi Regulagedda, Zackary Leech

Text pre-processing was performed on the dataset, following which it was vectorized by using CountVectorizer from sci-kit learn. Following this the vectors were scaled using StandardScaler and then three different basic classification models were tested - Logistic Regression, Naive Bayes and Random Forest. Their base model accuracy scores are given below -

```
1  from sklearn.metrics import accuracy_score
2  print("Naive Bayes", accuracy_score(y_test, y_pred))
3  print("Random Forest", accuracy_score(y_test, y_pred_2))
4  print("Logistic Regression", accuracy_score(y_test, y_pred_3))
✓ 0.0s

Naive Bayes 0.6860779020439646
Random Forest 0.8177786347859622
Logistic Regression 0.8069803316621674
```

Figure 1: Result showing the baseline accuracy of 3 different algorithms

Following this, a cross-validation search was performed on the ranges of hyper parameters for Random Forest since it had the highest initial accuracy. The ranges chosen are below

```
▾ RandomizedSearchCV
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(n_jobs=3), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'criterion': ['gini', 'entropy',
                                                      'log_loss'],
                                        'min_samples_split': [2, 3, 4, 5],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
    ▾ estimator: RandomForestClassifier
    RandomForestClassifier(n_jobs=3)
        ▾ RandomForestClassifier
        RandomForestClassifier(n_jobs=3)
```

Figure 2: Range of hyperparameter values for CV search

We used a 3 fold cross validation over 100 searches making a total of 300 models fit. This provided the following values for the hyperparameters -

```
1  rf_random.best_params_
✓ 0.0s

{'n_estimators': 1600,
 'min_samples_split': 2,
 'criterion': 'gini',
 'bootstrap': False}
```

Figure 3: Range of hyperparameter values for CV search

However, many of the "optimal" values provided are just the default values of the parameters. Specifically, *min_samples_split* which is the minimum number of positive/negative examples required to split a leaf, *criterion* which is the way information at each node is calculated and *bootstrap*, which is whether to use a bootstrapped dataset of the entire set are all at their default values. Therefore, the increase in accuracy was just 1%. The final accuracy is shown below -

Figure 4: Accuracy with best hyperparameters

Appendix

The code for reproducing our experiments is below

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV


def create_dataset(file_path):
    file = open(file_path)  # opening the dataset
    # these store the different data items in each line
    y = []
    X = []
    while True:
        tweet = (
            file.readline().rstrip()
        )  # removing excess whitespace, endline chars at the end of each line
        split = tweet.split(" ")  # splitting into list of words at space
        y.append(split[-1])  # last work of each sentence is the target
        words = split[:-1]  #
        if words:
            sentence = " ".join(words[2:])
        X.append(sentence)

        if not tweet:
            break
    return X, y


def convert_to_dataframe(X):
    df = pd.DataFrame(X, columns=["tweet_text"])
    return df


# processing data
X_list, y_list = create_dataset("waseemDataSet.txt")
X = convert_to_dataframe(X_list)
```

```
tweet_text = X["tweet_text"]
X_train, X_test, y_train, y_test = train_test_split(
    tweet_text, y_list, test_size=0.33, random_state=42
)
# using character n-grams
vectorizer = CountVectorizer(
    ngram_range=(1, 3), token_pattern=r"(?u)\b\w+\b", analyzer="char"
)
scaler = StandardScaler(with_mean=False)
# modifying the data to vectorize and scale
X_train_counts = vectorizer.fit_transform(X_train)
X_train_scaled = scaler.fit_transform(X_train_counts)

# different clasifiers
clf = MultinomialNB().fit(X_train_scaled, y_train)
clf2 = RandomForestClassifier().fit(X_train_scaled, y_train)
clf3 = LogisticRegression(solver="liblinear").fit(X_train_scaled, y_train)

# fitting test data based on train metrics
X_test_counts = vectorizer.transform(X_test)
X_test_scaled = scaler.transform(X_test_counts)

# predictions
y_pred = clf.predict(X_test_counts)
y_pred_2 = clf2.predict(X_test_scaled)
y_pred_3 = clf3.predict(X_test_scaled)

# accuracy
print("Naive Bayes", accuracy_score(y_test, y_pred))
print("Random Forest", accuracy_score(y_test, y_pred_2))
print("Logistic Regression", accuracy_score(y_test, y_pred_3))

# Random Search CV
# specifying the params
rf = RandomForestClassifier(n_jobs=3)  # 3 jobs for parallelism
# ranges for hyperparams
n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
min_samples_split = [2, 3, 4, 5]
criterion = ["gini", "entropy", "log_loss"]
bootstrap = [True, False]

random_grid = {
    "n_estimators": n_estimators,
    "min_samples_split": min_samples_split,
    "bootstrap": bootstrap,
    "criterion": criterion,
}

rf_random = RandomizedSearchCV(
    estimator=rf,
    param_distributions=random_grid,
    n_iter=100,
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1,
)
# Fit the random search model - takes one hour
rf_random.fit(X_train_counts, y_train)
# using the best params
rf_best = RandomForestClassifier(
```

```
    n_estimators=1600, min_samples_split=2, criterion="gini", bootstrap=False
).fit(X_train_scaled, y_train)
# best accuracy
y_pred_rf = rf_best.predict(X_test_scaled)
print("Best hyperparameters accuracy: ", accuracy_score(y_pred_rf, y_test))
```