```python
import nltk
import pandas as pd
import numpy as np
import random
import re
import joblib
import json
from collections import defaultdict
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score


def preprocess_data(csvfile):
    """
    Read and preprocess CSV file complaint data

    """

    stops = stopwords.words('english')

    df = pd.read_csv(csvfile)

    ids = np.array(df["complaint_id"])
    product_groups = np.array(df["product_group"])
    raw_text_list = np.array(df["text"])

    clean_text_list = raw_text_list

    return ids, product_groups, clean_text_list

def define_group_labels(product_groups):
    """
    Map product group text label to numerical labels (random assignment) and return diction

    """

    group_to_label = dict()
    label_to_group = dict()
    for idx, g in enumerate(set(product_groups)):
        group_to_label[g] = idx
        label_to_group[idx] = g

    return group_to_label, label_to_group

def create_vectorizer(document_list, vocabulary=None):
    """
    Function to create a vectorizer for input

    """
    vectorizer = CountVectorizer(ngram_range=(1,2), vocabulary=vocabulary, max_features=500
    fit_vectorizer = vectorizer.fit(document_list)

    return fit_vectorizer

def vectorize_documents(document_list, vectorizer):
    """
    Function to vectorize input text documents

    """
    X = vectorizer.transform(document_list)

    return X

```

```
def create_balanced_binary_sample(document_list, labels, focus_label):
    """
    Create equal, binary test data for One-Versus_Rest classification

    """
    target_documents = []
    target_labels = [] # Class 1
    other_documents = []
    other_labels = [] # All other classes mapped to Class 0

    for i in range(len(document_list)):
        if labels[i] == focus_label:
            target_documents.append(document_list[i])
            target_labels.append(1)
        else:
            other_documents.append(document_list[i])
            other_labels.append(0)

    to_fill = len(target_documents)

    # Sample other classes randomly
    ziplist = list(zip(other_documents, other_labels))
    sampled_others = random.sample(ziplist, to_fill)
    second_documents, second_labels = zip(*sampled_others)

    # Store in lists
    output_documents = [d for d in target_documents]
    output_documents.extend([d for d in second_documents])
    output_labels = [l for l in target_labels]
    output_labels.extend([l for l in second_labels])

    return output_documents, output_labels

def create_kfolds(document_list, labels, k):
    """
    Randomize and split dataset into K folds (close but not exactly equal size folds)
    (NOTE: Unused for final model training)

    """
    x_folds = []
    y_folds = []

    combined_lists = list(zip(document_list, labels))
    random.shuffle(combined_lists)
    rand_documents, rand_labels = zip(*combined_lists)

    slicesize = len(rand_documents) // k

    for num in range(k):
        if num != k-1:
            x_folds.append(rand_documents[num*slicesize:(num+1)*slicesize])
            y_folds.append(rand_documents[num*slicesize:(num+1)*slicesize])
        else:
            x_folds.append(rand_documents[num*slicesize:])
            y_folds.append(rand_documents[num*slicesize:])

    return x_folds, y_folds

def elasticNet_feature_selection(x_folds, y_folds, vectorizer):
    """
    Function to train One-Versus-Rest Elastic Net Logistic Regression to find informative f
    (NOTE: Unused for final model training)

    """

    clf  = SGDClassifier(loss="log",penalty="elasticnet")

    for fold_idx in range(len(x_folds)):

        test_documents = x_folds[fold_idx]
        test_labels = y_folds[fold_idx]

        train_documents = []
        train_labels = []
        for idx in range(len(x_folds)):
            if idx != fold_idx:
                train_documents.extend(x_folds[idx])
                train_labels.extend(y_folds[idx])

        X_train = vectorize_documents(train_documents, vectorizer)
        y_train = np.array(train_labels)

        print(X_train.shape)
        print(len(y_train))

        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_new)
        model_accuracy = accuracy_score(y_pred, y_new)

        print("SGD Model Accuracy on Fold %d: " %fold_idx)
        print(model_accuracy)

    return best_model

if __name__ == "__main__":

    # Preprocess data
    print("\nPreprocessing documents...")
    ids, product_groups, clean_text_list = preprocess_data("case_study_data.csv")

    # Define numerical target labels (y)
    group_to_label, label_to_group = define_group_labels(product_groups)
    group_numbers = [group_to_label[group] for group in product_groups]

    # Create initial vectorizer
    print("\nCreating Vectorizer...")
    firstvectorizer = create_vectorizer(clean_text_list)

    for group, label in group_to_label.items():

        binary_docs, binary_labels = create_balanced_binary_sample(clean_text_list, group_n

        print(label, group)
        print(len(binary_docs))

        #x_folds, y_folds = create_kfolds(binary_docs, binary_labels, 5)

        #elasticNet_feature_selection(x_folds, y_folds, firstvectorizer)

        traintest_split = int(len(binary_docs)*.8)

        X_train = vectorize_documents(binary_docs[0:traintest_split], firstvectorizer)
        y_train = np.array(binary_labels[0:traintest_split])

        X_test = vectorize_documents(binary_docs[traintest_split:], firstvectorizer)
        y_test = np.array(binary_labels[traintest_split:])

        clf  = SGDClassifier(loss="log",penalty="elasticnet")

        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        print("SGD Model Accuracy: ")
        print(accuracy_score(y_pred, y_test))

        idx_to_feature = dict()
        feature_to_weights = defaultdict(list)
        selected_vocabulary = set()

        for key, value in firstvectorizer.vocabulary_.items():
            idx_to_feature[value] = key
        for idx, c in enumerate(clf.coef_[0]):
            if abs(c) > 0:
                selected_vocabulary.add(idx_to_feature[idx])

    print("\nVocabulary Size: %d" %len(selected_vocabulary))
    print("Creating Fine Tuned Vectorizer...")
    newvectorizer = create_vectorizer(clean_text_list, vocabulary=selected_vocabulary)

    splitidx = int(len(ids) * .8)

    combined_lists = list(zip(product_groups, clean_text_list))
    random.shuffle(combined_lists)
    rand_product_groups, rand_text_list = zip(*combined_lists)


    print("\nVectorizing documents...")
    X = vectorize_documents(rand_text_list[0:splitidx], newvectorizer)
    y = np.array([group_to_label[group] for group in rand_product_groups[0:splitidx]])

    print("\nVectorizing documents...")
    X_new = vectorize_documents(rand_text_list[splitidx:], newvectorizer)
    y_new = np.array([group_to_label[group] for group in rand_product_groups[splitidx:]])

    alphas = [.001, .01, .05, .5, .75, 1]

    best_alpha = .05
    current_score = 0
    for a in alphas:
        clf  = MultinomialNB(alpha=a)

        clf.fit(X, y)
        y_pred = clf.predict(X_new)

        acc = accuracy_score(y_pred, y_new)

        print("Model With Alpha %f: " %a)
        print(acc)
        if acc > current_score:
            current_score = acc
            best_alpha = a

    print("\nVectorizing Final documents...")
    X_final = vectorize_documents(rand_text_list, newvectorizer)
    y_final = np.array([group_to_label[group] for group in rand_product_groups])

    final_model = MultinomialNB(alpha=best_alpha)
    final_model.fit(X_final, y_final)

    joblib.dump(newvectorizer, "NWCaseStudyvectorizer.joblib")
    joblib.dump(final_model, "NWCaseStudyNBModel.joblib")

    with open("group_mapping.json", "w") as file:
        file.write(json.dumps(group_to_label))
```