PREDICTING THE SENTIMENT

Objective

The objective of this project is to classify the sentiment of sentences from movie, restaurant, or product reviews into positive or negative categories. Specifically, the project aims to:

1. **Preprocess** the text data to prepare it for modeling.
2. **Apply a Support Vector Machine (SVM) classifier** to predict sentiment based on the processed text data.
3. **Evaluate the classifier performance** using various values of the hyperparameter "C" and through k-fold cross-validation.

Summary

The dataset for this project consists of 3000 labeled sentences, with 2500 sentences allocated for training and 500 for testing. The sentences are labeled as either positive (1) or negative (0). This project employs a Support Vector Machine (SVM) to predict sentiment.

1. **Data Loading and Preprocessing**:
   - Loaded the dataset from a text file and split sentences and labels.
   - Transformed labels from binary (0,1) to (-1,1) for SVM compatibility.
   - Removed digits and punctuation, converted text to lowercase, and removed common stop words.
   - Used CountVectorizer to convert the text data into a bag-of-words representation.
   - Split the data into training and testing sets.
2. **Model Training**:
   - Implemented the SVM using the LinearSVC class from scikit-learn with different values for the regularization parameter "C".
   - Evaluated the performance of the SVM classifier on the training and testing datasets for each value of "C".
3. **Hyperparameter Tuning**:
   - Tested different values of "C" to determine the optimal regularization strength.
   - Used k-fold cross-validation to estimate the error associated with each value of "C".

Results

1. **Model Performance**:
   - Evaluated the SVM classifier for several values of "C" (0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0).

- Found that the minimum test error was achieved with C=1.0C = 1.0C=1.0, indicating this value provides the best balance between training and generalization error for this dataset.

2. **Cross-Validation**:
   - Performed k-fold cross-validation to estimate the error for C=1.0C = 1.0C=1.0 across different values of k (from 2 to 9).
   - The cross-validation results provide a more robust estimate of the classifier's performance and help in understanding the variability of the model's error with different partitions of the training data.

Conclusion

The Support Vector Machine classifier effectively predicted sentiment with a reasonably low test error rate. The optimal regularization parameter CCC was found to be 1.0, balancing model complexity and generalization. Cross-validation further confirmed the stability and reliability of this parameter choice.

Code :

```
'''

We will be predicting the sentiment (positive or negative) of a single sentence taken from a review of a

movie, restaurant, or product. The data set consists of 3000 labeled sentences, which we divide into a

training set of size 2500 and a test set of size 500. We have already used a logistic regression classifier. Now,

we will use a support vector machine.

'''


import string

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

matplotlib.rc('xtick', labelsize=14)

matplotlib.rc('ytick', labelsize=14)

from sklearn.feature_extraction.text import CountVectorizer

from sklearn import svm

#Loading and preprocessing the data

#Read in the data set:

with open("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\PROPER DATASETS Non Edited\\ML Projects and Datasets\\data\\full_set.txt") as f:

 content = f.readlines()

#Remove leading and trailing white space:

content = [x.strip() for x in content]

#Separate the sentences from the labels:

sentences = [x.split("\t")[0] for x in content]

labels = [x.split("\t")[1] for x in content]


#Transform the labels from '0 versus 1' to '-1 versus 1':

y = np.array(labels, dtype='int8')
```

```
y = 2*y - 1


#Let us define a function, "full_remove" that takes a string x
# and a list of characters from a removal_list and returns with all the
# characters in removal_list replaced by ' '.
def full_remove(x, removal_list):
    for w in removal_list:
        x = x.replace(w, ' ')
    return x


#Remove digits:
digits = [str(x) for x in range(10)]
digit_less = [full_remove(x, digits) for x in sentences]
#Remove punctuation:
punc_less = [full_remove(x, list(string.punctuation)) for x in digit_less]
#Make everything lower-case:
sents_lower = [x.lower() for x in punc_less]
#Define our stop words:
stop_set = set(['the', 'a', 'an', 'i', 'he', 'she', 'they', 'to', 'of', 'it', 'from'])
#Remove stop words
sents_split = [x.split() for x in sents_lower]
sents_processed = [" ".join(list(filter(lambda a: a not in stop_set, x))) for x in sents_split]


#Transform to bag of words representation:
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None,
preprocessor = None, stop_words = None, max_features = 4500)
data_features = vectorizer.fit_transform(sents_processed)
#Append '1' to the end of each vector.
data_mat = data_features.toarray()
#Split the data into testing and training sets:
np.random.seed(0)
```

```
test_inds = np.append(np.random.choice((np.where(y==-1))[0], 250,

replace=False), np.random.choice((np.where(y==1))[0], 250,

replace=False))

train_inds = list(set(range(len(labels))) - set(test_inds))

train_data = data_mat[train_inds,]

train_labels = y[train_inds]

test_data = data_mat[test_inds,]

test_labels = y[test_inds]

print("train data: ", train_data.shape)

print("test data: ", test_data.shape)


'''

Fitting a support vector machine to the data :


In support vector machines, we are given a set of examples ( 1, 1),…,( , ) and we want to find a weight

vector w∈ℝd that solves the following optimization problem:

scikit-learn provides an SVM solver that we will use. The following routine takes the constant "C" as
an input

(from the above optimization problem) and returns the training and test error of the resulting SVM
model. It

is invoked as follows:

 "training_error, test_error = fit_classifier(C)"

 The default value for parameter "C" is 1.0.

'''


#Fitting a support vector machine to the data

def fit_classifier(C_value=1.0):

   clf = svm.LinearSVC(C=C_value, loss='hinge')

   clf.fit(train_data,train_labels)

   # Get predictions on training data

   train_preds = clf.predict(train_data)

   train_error = float(np.sum((train_preds > 0.0) != (train_labels > 0.0)))/len(train_labels)
```

```
    # Get predictions on test data
    test_preds = clf.predict(test_data)
    test_error = float(np.sum((test_preds > 0.0) != (test_labels > 0.0)))/len(test_labels)
    return train_error, test_error
cvals = [0.01,0.1,1.0,10.0,100.0,1000.0,10000.0]
for c in cvals:
    train_error, test_error = fit_classifier(c)
    print ("Error rate for C = %0.2f: train %0.3f test %0.3f" % (c, train_error, test_error))
#We see that the minimum test error is for C = 1.0.


'''
We see that the minimum test error is for C = 1.0.
3. Evaluating C by k-fold cross-validation
 As we can see, the choice of "C" has a very significant effect on the performance of the SVM
classifier. We were able to assess this because we have a separate test set. In general, however, this
is a luxury we won't possess.
 A reasonable way to estimate the error associated with a specific value of "C" is by k-fold cross
validation:
o Partition the training set into "k" equal-sized sized subsets.
o For i=1,2,...,k, train a classifier with parameter C..
 Average the errors: "(e_1 + ... + e_k)/k"
 The following procedure, cross_validation_error, does exactly this. It takes as input:
o the training set "x,y"
o the value of "C" to be evaluated
o the integer "k"
 It returns the estimated error of the classifier for that particular setting of "C"
'''


#3. Evaluating C by k-fold cross-validation
def cross_validation_error(x,y,C_value,k):
    n = len(y)
    # Randomly shuffle indices
    indices = np.random.permutation(n)
    # Initialize error
    err = 0.0
    # Iterate over partitions
    for i in range(k):
        # Partition indices
        test_indices = indices[int(i*(n/k)):int((i+1)*(n/k) - 1)]
        train_indices = np.setdiff1d(indices, test_indices)
    # Train classifier with parameter c
    clf = svm.LinearSVC(C=C_value, loss='hinge')
    clf.fit(x[train_indices], y[train_indices])
    # Get predictions on test partition
    preds = clf.predict(x[test_indices])
    # Compute error
    err += float(np.sum((preds > 0.0) != (y[test_indices] > 0.0)))/len(test_indices)
    return err/k
#Let us print out the eoees or ifferen vaues of k
for k in range(2,10):
    print("cross_validation_error: ")
    print(cross_validation_error(train_data,train_labels,1.0,k))
```