2η Εργασία Τεχνητής Νοημοσύνης

Γιώργος Λεβέντης(3130114)

Αρχική Επεξεργασία

Το μοντέλο που επιλέχθηκε είναι η λογιστική παλινδρόμηση με στοχαστική ανάβαση κλίσης και όρο κανονικοποίησης στην αντικειμενική συνάρτηση πάνω στο σύνολο δεδομένων Enron-Spam.

Αρχικά διαβάζουμε τα δεδομένα από τα αρχεία που έχουμε κατεβάσει (σε raw μορφή) και τα φορτώνουμε σε ενα Dataframe με την επιπλέον κατηγορία ‘class’ (1=’spam’ , 0=’ham’).Η συγκεκριμένη κατηγορία θα αποτελέσει και τον πίνακα αποτελεσμάτων μας στην εκπαίδευση του μοντέλου.

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.sparse as sp
import LogisticRegression

NEWLINE = '\n'
SKIP_FILES = {'cmds'}

def read_files(path):
    for root,dir_names,file_names in os.walk(path):
        for path in dir_names:
            read_files(os.path.join(root,path))
        for file_name in file_names:
            if file_name not in SKIP_FILES:
                file_path =os.path.join(root,file_name)
                if os.path.isfile(file_path):
                    past_header,lines= False,[]
                    f = open(file_path,encoding='latin-1')
                    for line in f :
                        if past_header:
                            lines.append(line)
                        elif line == NEWLINE:
                            past_header = True
                    f.close()
                    content = NEWLINE.join(lines)
                    yield file_path , content

def build_data_frame(path , labeling):
    rows= []
    index = []
    for file_name , text in read_files(path):
        rows.append({'text' : text , 'class' : labeling})
        index.append(file_name)
    df = pd.DataFrame(rows,index=index)
    return df

HAM = 0
SPAM = 1

SOURCES = [('beck-s', HAM),
           ('farmer-d' , HAM),
           ('kaminski-v', HAM),
           ('kitchen-l', HAM),
           ('lokay-m', HAM),
           ('williams-w3', HAM),
           ('BG' , SPAM),
           ('GP' , SPAM),
           ('SH' , SPAM)]

df = pd.DataFrame({'text': [] ,'class': []})
for path, labeling in SOURCES:
    df = df.append(build_data_frame(path , labeling))
df = df.reindex(np.random.permutation(df.index))
```

Έπειτα,για την δημιουργία κατηγοριών ώστε να εκπαιδεύσουμε το μοντέλο, χρησιμοποιούμε την κλάση CountVectorizer(min_df=0.1) του πακέτου sklearn.feature_extraction.text.Η συγκεκριμένη κλάση διαβάζει όλα τα e-mail στο Dataframe και κάθε λέξη που υπάρχει περισσότερο από 10% στο σύνολο των λέξεων την προσθέτει ως διάνυσμα στον πίνακα με τα features.

Επειδή η συγκεκριμένη μέθοδος επιστρέφει έναν αρκετά αραιό πίνακα (sparse matrix) χρησιμοποιούμε την μέθοδο scipy.todense() για να επαναφέρουμε το X σε κανονικό πίνακα.

```
55 df = df.reindex(np.random.permutation(df.index))
56
57 from sklearn.feature_extraction.text import CountVectorizer
58 word_count= CountVectorizer(min_df=0.1)
59 X = word_count.fit_transform(df['text'])
60 X = sp.csc_matrix.todense(X)
```

Η Κλάση LogisticRegression()

```
1 import numpy as np
2
3 class LogisticRegression :
4
5     def __init__(self , learning_rate = 0.001, epochs=140 , reg_term=0.0001 , bias = True , batch_size=64):
6         self.learning_rate = learning_rate
7         self.epochs = epochs
8         self.reg_term = reg_term
9         self.bias = True
0         self.batch_size = batch_size
1         print('Initialized Logistic Regression object')
```

Η μέθοδος __init__ είναι η αρχικοποίηση των μεταβλητών της κλάσης

```
def add_bias(self, X):
    bias = np.ones((X.shape[0],1))
    return np.concatenate((bias,X),axis=1)
```

Η μέθοδος add_bias παίρνει σαν όρισμα έναν πίνακα και προσθέτει μια στήλη απο 1 στην αρχή του.Αυτό είναι αναγκαίο για την λογιστική παλινδρόμηση.

```
def sigmoid(self,X , weight):
    z = X.dot(weight.T)
    result = 1/(1.0+np.exp(-z))
    result[result == 1 ] = 0.9999999
    result[result == 0 ] = 0.0000001
    return result
```

Η μέθοδος sigmoid είναι σιγμοειδής συνάρτηση,με έναν επιπλέον έλεγχο ώστε να μην επιστρέφεται 1 η 0 λόγο υπερχείλισης.

```
def log_likelihood(self , X,y,weight):
    term1 = y.T*np.log(self.sigmoid(X,weight))
    #DEBUG print('term1 shape :',term1.shape)
    term2 = (1-y).T*np.log(1-self.sigmoid(X,weight))
    #DEBUG print('term2 shape :', term2.shape)
    term3 = (0.5*self.reg_term*(weight.dot(weight.T)))
    #DEBUG print('term3 shape :', term3.shape)
    result = np.sum(term1+term2) - term3
    #DEBUG print('log likelihood shape :', result.shape)
    return result
```

Η μέθοδος log_likelihood υπολογίζει την λογαριθμική πιθανοφάνεια της λογιστικής παλινδρόμησης χρησιμοποιώντας επίσης και παράγοντα κανονικοποίησης λ .

```
def gradient(self , X , y ,weight ):
    result = X.T.dot((y - self.sigmoid(X,weight))) - (self.reg_term*weight).T
    #DEBUG print('gradient shape: ', result.shape)
    return result.T
```

Η μέθοδος gradient υπολογίζει την κλίση της συνάρτησης για τον αλγόριθμος της στοχαστικής ανάβασης κλίσης.

```
def fit(self ,X ,y):
    print('Starting fit function')
    if self.bias :
        X = self.add_bias(X)
    weight = np.random.rand(1,X.shape[1])
    np.random.seed(seed=1)
    indices = np.random.permutation(len(X))
    X = X[indices,:]
    y = y[indices]
    cost_history = np.zeros(self.epochs)
    for i in range(0,self.epochs):
        for j in range(0,X.shape[0],self.batch_size):
            X_batch = X[i:i+self.batch_size,:]
            y_batch = y[i:i+self.batch_size]
            weight = weight + self.gradient(X_batch,y_batch,weight)*self.learning_rate
        loss = self.log_likelihood(X_batch , y_batch , weight)
        cost_history[i] = loss
    return weight , cost_history
```

Η μέθοδος fit είναι ο ‘πυρήνας’ της λογιστικής παλινδρόμησης.Αρχικά λαμβάνει ένα X και ανάλογα το flag bias καλεί την συνάρτηση add_bias.Ύστερα υλοποιεί τον αλγόριθμο στοχαστικής ανάβασης κλίσης με mini batches για εξοικονόμηση χρόνου σε σχέση με τον κλασικό αλγόριθμο στοχαστικής ανάβασης κλίσης.Το κάθε mini batch επιλέγεται τυχαία για να εξασφαλιστεί καλύτερη ενημέρωση των βαρών της συνάρτησης.Επιπλέον υπολογίζεται σε κάθε ‘εποχή’ και το loss της συνάρτησης για να παρατηρήσουμε αν υπάρχει όντως βελτίωση μετά από κάθε επανάληψη.

```
def predict(self, X ,weight , bias=True) :
    if bias:
        X=self.add_bias(X)
    return ( np.round(self.sigmoid(X , weight)))
```

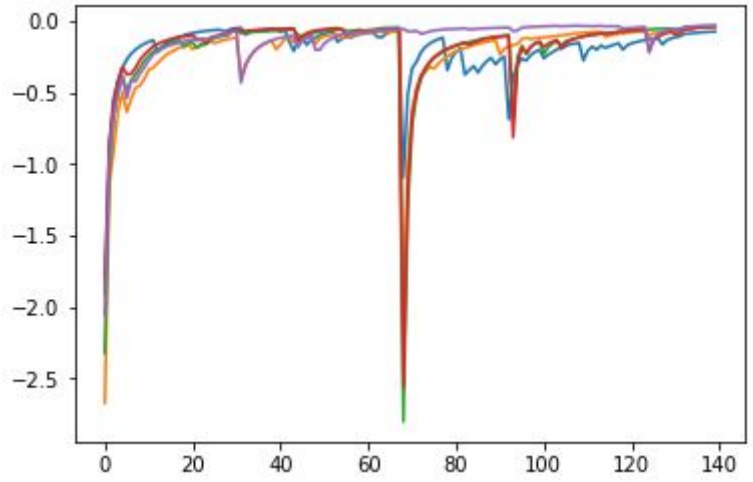
Τέλος , η συνάρτηση predict προβλέπει έναν πίνακα αποτελεσμάτων , χρησιμοποιώντας έναν πίνακα features X και τα ενημερωμένα βάρη της συνάρτησης.

Αποτελέσματα

```
clf = LogisticRegression.LogisticRegression()
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5)
i=0
for train_index,test_index in skf.split(X,y):
    i += 1
    print('Training ',i,'fold of ',skf.n_splits)
    X_train,X_test = X[train_index],X[test_index]
    y_train , y_test = y[train_index] , y[test_index]
    weight , log_likelihood = clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test , weight)
    print('Accuracy is ' , (y_test==y_pred).mean())
    from sklearn.metrics import f1_score
    print('f1_micro is ', f1_score(y_test,y_pred))
    plt.plot(log_likelihood)
```

Για την αποτελεσματικότερη εκπαίδευση του αλγορίθμου χρησιμοποιείται η κλάση StratifiedKFolds του πακέτου sklearn.model_selection.Η συγκεκριμένη κλάση χωρίζει το δείγμα σε διαφορετικούς υποπίνακες και εκπαιδεύει το μοντέλο ξεχωριστά,υπολογίζοντας ενδιάμεσα την ευστοχία και το f1_micro score.

```
Training  1 fold of  5
Starting fit function
Accuracy is  0.9408153628207147
f1_micro is  0.9353729804056377
Training  2 fold of  5
Starting fit function
Accuracy is  0.9329345088161209
f1_micro is  0.9241452991452992
Training  3 fold of  5
Starting fit function
Accuracy is  0.9282002834199339
f1_micro is  0.9203354297693921
Training  4 fold of  5
Starting fit function
Accuracy is  0.9343410486537553
f1_micro is  0.9271615720524018
Training  5 fold of  5
Starting fit function
Accuracy is  0.9261533616753267
f1_micro is  0.9179065289690181
```



Τέλος βλέπουμε το διάγραμμα για την λογαριθμική πιθανοφάνεια της συνάρτησης.Παρατηρούμε ότι είναι συνηθισμένη σύμφωνα με την στοχαστική ανάβαση κλίσης και οτι είναι σχετικά ίδια για κάθε υποπίνακα εκπαίδευσης που χρησιμοποιούμε. Άρα αυτό σημαίνει οτι ο αλγόριθμος είναι αρκετά στοιβαρός στα δεδομένα εκπαίδευσης (robust).