به نام خداوند بخشنده مهربان

پروژه پاياني

يادگيري تحت نظارت

اعضاي گروه :

امير محمد آقاجاني – ۴۰۰۵۲۱۰۶۳

سيد محمد علي ميرمحمدي – ۴۰۰۵۲۲۲۶۵

بخش اول : توضيح روند كلي

در ابتدا به توضيح كلي الگوريتم در حالت سه كلاسه مي پردازيم. در حالت سه كلاسه هر كلمه در هر توئيت با توجه به برچسب توئيت شمرده مي شود يعني براي مثال داريم كه كلمه not ۱۰ بار در كلاس مثبت، ۱۰۰ بار در كلاس منفي و ۵ بار در كلاس خنثي تكرار شده است. حال با استفاده از naïve bayes (بدون محاسبه آلفا كه در داك آورده شده است) طبق فرمول احتمال پيشين (احتمال پيشين منظور اين است كه به طور براي كلاس خنثي بدون در نظر گرفتن هيچ گونه ويژگي صرفا با توجه به تعداد توئيت هاي خنثي و تعداد كل توئيت ها چقدر احتمال دارد كه اين توئيت خنثي باشد) ضرب در احتمال رخ دادن اين كلمه به شرط احتمال پيشين به ازاي همه كلمات موجود در توئيت ميتوانيم احتمال اين كه اين توئيت از جنس احتمال پيشين باشد را نشان دهد. در واقع به طور مثال احتمال منفي بودن توئيت با توجه به تعداد تكرار كلمات آن در كلاس منفي ها و تعداد تكرار آن ها در كل را محاسبه كرده و براي دو كلاس ديگر نيز اين احتمال را محاسبه ميكنيم. حال هر احتمالي كه بيشتر باشد را به عنوان پيش بيني بر گردانيم.

بخش دوم : توضيح كد

ابتدا از فايل template شروع ميكنيم. پس از كانستراكتور تابع train قرار دارد كه به ازاي هر توئيت تعداد توئيت ها و به ازاي كلمات آنها تعداد كلمات متناسب با label هر توئيت را update ميكند و اين كار را به ازاي تمام توئيت ها انجام ميدهد. تابع calculate_prior به ازاي هر كلاس صرفا بر اساس تعداد توئيت هاي آن كلاس احتمال رخداد آن كلاس را بررسي ميكند. تابع calculate_likelihood بر اساس تعداد كلمات در هر كلاس احتمال رخداد آن كلمه به شرط اينكه توئيت مورد نظر از كلاس خاصي باشد را محاسبه ميكند.

تابع classify نيز در مواجهه با هر توئيت به ازاي تمام كلاس ها احتمال اينكه اين توئيت متعلق به آن كلاس ها باشد را محاسبه كرده و احتمال ماكزيمم را به عنوان خروجي بر مي گرداند.

```
def train(self, data):
    # training process:
    # inputs: data(list) --> each item of list is a tuple
    # the first index of the tuple is a list of words and the second index is the label(positive, negative, or neutral)
    self.tweet_counts = len(data)
    for features, label in data:
        self.class_counts[label] += 1
        for feature in features:
            self.class_word_counts[label] += 1
            if (feature, label) in self.vocab.keys():
                self.vocab[(feature, label)] += 1
            else:
                self.vocab[(feature, label)] = 1

def calculate_prior(self, label):
    # calculate log prior
    # you can add some attributes to this method

    return log10(self.class_counts[label] / self.tweet_counts)

def calculate_likelihood(self, word, label):
    # calculate likelihhood: P(word | label)
    # return the corresponding value

    if (word, label) in self.vocab.keys():
        return log10((self.vocab[(word, label)] + 1) / (self.class_word_counts[label] + 3))
    return log10(1 / (self.class_word_counts[label] + 3))

def classify(self, features):
    # predict the class
    # inputs: features(list) --> words of a tweet

    what_class = {'negative': 0, 'neutral': 0, 'positive': 0}

    for label in self.classes:
        for feature in features:
            what_class[label] += self.calculate_likelihood(feature, label)
        what_class[label] += self.calculate_prior(label)

    best_class = max(what_class, key=what_class.get)

    return best_class
```

حال به بررسي فايل run مي پردازيم. در ابتدا تابع preprocess يك رشته (متن توئيت) را دريافت كرده آن را تميز كرده و كاراكتر هاي اضافي را از آن حذف ميكنيم.سپس كلماتي كه معني مثبت يا منفي ندارند مثل حروف ربط را حذف كرده و خروجي را به صورت ليستي از كلمات بر ميگردانيم.

تابع load_data داده را از فايل csv خوانده و در نهايت به صورت ليستي از tuple هاي دوتايي (كلمات توئيت و label) بر مي گرداند.

تابع load_test_data همين كار را براي داده هاي تست انجام مي دهد. سپس در بخش بعد train انجام شده و زمان آن ثبت مي شود و در بخش دوم validation انجام شده و دقت چاپ مي شود و در بخش آخر نيز label گذاري صورت ميگيرد.

```
def preprocess(tweet_string):
    # clean the data and tokenize it
    str = re.sub(r"[&!\"#$%&()*+-./:;<=>?@[\]^_{|}~\n -' 0123456789\\]"," ", tweet_string)
    str = unidecode.unidecode(str)
    strs = str.split(' ')
    features = []
    stemmer = PorterStemmer()
    features = list(stemmer.stem(word) for word in strs)
    features = list(filter(None , features))
    li = ["to", "of" , "as" , "a" , "an" , "with" , "for" , "and" , "or" , "my", "your" , "his", "her", "ours", "their", "me", "him", "us" , "them", "
    final_features = []
    for word in features:
        if not (word in li or word.isnumeric()) and len(word) > 1:
            final_features.append(word)
    return final_features


def load_data(data_path):
    # load the train csv file and return the data
    grass_data = pd.read_csv(data_path)
    data = []
    for tweet in grass_data.iterrows():
        data.append((preprocess(str(tweet[1].text)), str(tweet[1].label_text)))
    return data


def load_test_data(data_path):

    grass_data = pd.read_csv(data_path)
    data = []
    for tweet in grass_data.iterrows():
        data.append(preprocess(str(tweet[1].text)))
    return data
```

```
# train your model and report the duration time
start = time()
train_data_path = "train_data.csv"
eval_data_path = "eval_data.csv"
test_data_path = "test_data_nolabel.csv"
classes = ['negative', 'neutral', 'positive']
nb_classifier = NaiveBayesClassifier(classes)
nb_classifier.train(load_data(train_data_path))
end = time()
print(f"training duration time:", round(end - start, 2), "second")


# check on evaluation data
eval_data = load_data(eval_data_path)
all_tweets = 0
correct_guess = 0
for eval_tweet, label in eval_data:
    all_tweets += 1
    guess_label = nb_classifier.classify(eval_tweet)
    if guess_label == label:
        correct_guess += 1
print(f"correctness percent on evaluation data: {round(correct_guess / all_tweets * 100, 2)}%")


# test on test data:
test_data = load_test_data(test_data_path)
number = 0
f = open("labels.txt", "a")
for test_tweet in test_data:
    guess_label = nb_classifier.classify(test_tweet)
    output = str(number) + ": " + guess_label + "\n"
    f.write(output)
    number += 1
f.close()
```

پایان