```
def load_data(trn_file_name, tst_file_name):
    df_trn = pd.read_csv(trn_file_name, header=None, sep="/")
    df_tst = pd.read_csv(tst_file_name, header=None, sep="/")
```

داده ها خوانده میشوند از داخل فایل.

جدا کننده داده ها همان / است که کلمات و tag ها را از هم جدا میکنه. داده های ترین و تست را یکجا خواندم.

```
words = df.iloc[:, 0].factorize()
tags = df.iloc[:, 1].factorize()
```

از تابع factorize استفاده کردم برای ستون های صفر و یک کلمات و تگ ها که این ها را کد گذاری کند یعنی بجای هر چیزی که داشتیم داخل کلمات 0،1،2،3... قرار میده. حتی اگه کلمات 1،2،3 هم باشند باز این کد گذاری انجام خواهد شد. باعث میشه در ادامه وقتی به اندیس یک آرایه اشاره میشه بتونیم راحت کار رو ادامه بدیم.

```
n_trn = df_trn.shape[0]
```

تعداد داده های آموزشی را از اینجا بدست میارم.

```
trn_words = words[0][:n_trn]
tst_words = words[0][n_trn:]

trn_tags = tags[0][:n_trn]
tst_tags = tags[0][n_trn:]
```

و دو باره داده ها هم ترین هم تست رو از هم جدا میکنم هم تگ ها و هم داده ها

```
words_ind = words[1]
tags_ind = tags[1]
```

اندیس گذاری انجام شده یعنی بجای هر کلمه ای چه اندیسی استفاده شده اینجا از همون خروجی تابع factorize جدا میکنم. این 2تا اندیس گذاری را نشان میدهند. مثلا بجای کلمه test چه اندیسی بکار رفته اینجا خود کلمات هستند و اندیسی داخل اون اندیس کلمات قرار داره نشون میده که کد اون کلمه ها رو نشون میده که داخل این trn_words و test_words و trn_tags و tst_tags استفاده شده.

```
return trn_words, trn_tags, tst_words, tst_tags, words_ind, tags_ind
```

و در نهایت خروجی را بر میگردونم.

```
def add_one_smoothing(words, tags, sharp, n_tags, n_words):
```

کاری که این تابع انجام میده 3 تا پارامتر رو باید تخمین بزنیم آرایه A و B و پی p

ورودی تابع words, tags, و sharp برای تعیین انتها کلمه چه کدی (###) استفاده شده. n_tags و n_words تعداد کلمات و تگ های متمایز.

```
p = np.zeros((n_tags,))
```

آرایه p را پیدا میکنم.

```
ind = np.where(tags == sharp)[0]
```

پیدا میکنم کجاها sharp است که بتونم مرز بین کلمات رو تشخیص بدم.

```
n_sentences = ind.shape[0]
```

تعداد sharp ها به ما تعداد جملات را هم خواهد گفت.

```
ind = np.insert(ind, 0, -1)[:-1] + 1
```

اندیس sharp هایی که پیدا کردم بشون یک اندیس -1 هم در ابتدا بشون اضافه میکنم و به همشون یک +1 اضافه میکنم. که الان اینا مرزهای بین جملات را نشون میدن شروع و انتهای جملات

```
p[i] = (np.count_nonzero(tags[ind] == i) + 1) / (n_sentences + n_tags)
```
در نهایت برای اینکه p را بدست بیارم میام در داخل tags های اون اندیس ها نگاه میکنم و دونه دونه اون تگهای خاص را میشمرم انیجا یه 1+ داریم و تعداد کل جملات + تعداد کل تگ ها هما فرمول تمرین استفاده شده.

```
a = np.zeros((n_tags, n_tags))
b = np.zeros((n_tags, n_words))
```
آرایه های a و b را ایجاد میکنیم که a تعدا تگها در تعداد تگها عنصر خواهد داشت و b تعداد تگها در تعداد کلمات عنصر داره.

```
for i in range(tags.shape[0] - 1):
    if tags[i] != sharp:
        a[tags[i], tags[i + 1]] += 1
        b[tags[i], words[i]] += 1
```
داخل آرایه تگ ها یک loop ایجاد میکنم.

و اگر تگی برابر با شارپ نباشد یعنی به انتهای جمله نرسیده باشیم.

قرار میدم همان فرمول Qt بابر با tag[i] باشه و Qt+1 برابر با tags[j] باشه و این رو + 1 خواهم کرد و در ادامه نرمال خواهم کرد.

برای b هم همان کار را خواهم کرد که در حالت i قرار داشته باشیم و کلمه ورودیمون در اون حالت کلمه i ام باشه مشاهده ما و تگمون هم برابر با tag[I] باشه و + 1 میکنیم.

```
c_q = a.sum(axis=1)
a = (a + 1) / (c_q[:, np.newaxis] + n_tags)

b = (b + 1) / (c_q[:, np.newaxis] + n_words)
return a, b, p
```
در نهایت هر دو آرایه را نرمالسازی روش انجام میدیم بر میگردونیم.

```
def viterbi(a, b, p, o):
    N = a.shape[0]
    T = o.shape[0]
```
الگوریتم Viterbi است که N و T را بدست آوردم.

```
vit = np.zeros((N, T))
back_pointer = np.zeros((N, T))
```
آرایه ها را ایجاد کردم.

```
vit[:, 0] = np.log(p) + np.log(b[:, o[0]])
```
فرمول اولیه Viterbi است. اینجا اندیس ها 0 شروع میشه اینجا برابر با 1 خواهد شد. معادل همون for است که تو الگوریتم نوشته شده.

```
back_pointer[:, 0] = 0
```
بک پوینتر s و صفر هم خواهد شد صفر.

```
for t in range(1, T):
    for s in range(N):
```
کاری که انجام میدیم 2 حلقه داشتیم T از 2 شروع میشد که به خاطر اندیس ها اینجا از 1 شروع میشه

و s هم از 1 تا N بود که اینجا از صفر تا n-1 طبیعتا خواهد بود.

```
a_max = np.argmax(vit[:, t - 1] + np.log(a[:, s]))
back_pointer[s, t] = a_max
vit[s, t] = np.log(b[s, o[t]]) + vit[a_max, t - 1] + np.log(a[a_max, s])
```

کاری که اونجا انجام داده بود هم max بدسته آورده بود هم argmax خب من خیلی ساده argmax رو اول بدست میارم بعد back_pointer را برابر با argmax قرار میدم.

و Viterbi[s,t] قرار میدم در واقع از a_max که بدست آوردم استفاده میکنم و مقدارش را حساب میکنم و ذخیره میکنم.

```
best_path_pointer = np.argmax(vit[:, T - 1])
```

این هم طبق تمرین میشه argmax(viterbi) اونجا برای s های مختلف حساب کرده بنابراین " : " خواهیم داشت یعنی تمام موارد و ستون آخر.

```
best_path = np.zeros((1,), dtype=int)
best_path[0] = best_path_pointer
```

best_path را اینجا ایجاد کردم باید داخلش یه دونه عنصر قرار داشته باشه و اون یه دونه عنصر را برابر با best_path_pointer قرار دادم. طبق چیزی که تو تمرین گفته شده.

```
for t in range(T - 1, 0, -1):
    best_path = np.insert(best_path, 0, back_pointer[best_path[0], t])
```

و دونه دونه برگشتم و مقادیر جدیدتر را insert کردم داخل best_path ابتداش صفر کردم و هی برمیگردیم از best_path که هر لحظه پیدا کردیم اون رو اندیس اولیه آرایه back_pointer قرار میدیم و T رو هم از

T- 1 تغییر میدم تا مقدار 1 به خود صفر نخواهد رسید. و این انتخابهایی که داشتیم a_max ها را داخل این best_path ، insert میکنیم. و در نهایت best_path را بر میگردانیم.

```
def apply_viterbi(a, b, p, tst_words, tags_ind, words_ind, word_sharp):
    ind = np.where(tst_words == word_sharp)[0]
    n_sentences = ind.shape[0]
    ind = np.insert(ind, 0, -1) + 1
```

آرایه apply_viterbi : a,b,p که از smooth بدست آوردیم را میگیره کلمات تست و تگ اندکس و word_ind را میگیره و چه کدی تو کلمات به sharp اختصاص داده شده میگیره.

بعد کاری که انجام میده مرز بین جملات را مشخص میکنه.

```
tags = []
for i in range(n_sentences):
    snt = tst_words[ind[i] : ind[i + 1]]
```

دونه دونه حملات را جدا میکنه

```
path = viterbi(a, b, p, snt)
```

و میده به الگوریتم Viterbi و مسیر را مشخص میکنه

```
    tags = tags + [tags_ind[j] for j in path]
return tags
```

و بعد گفته بودیم که تمام تگها کد گذاری شدن اینجا تگ ها را از کدگذاری خارجشون میکنیم.

```
tags = []
```

و در نهایت tags خواهد شد کدگذاری که برای هر کدام از جملات انجام دادیم. ودر نهایت tags ها رو بر میگردانیم.

```
data_sets = {
    "IC": ("ictrain.txt", "ictest.txt"),
    "IC2": ("ic2train.txt", "ic2test.txt"),
    "EN": ("entrain.txt", "entest.txt"),
    # "CZ": ("cztrain.txt", "cztest.txt"),
}
```

آرایه هایی داریم که به ما داده شده "CZ" کدگذاریش مشخص نبود برای خواندنش و جواب نداد.

```
for ds in data_sets:
    trn_file_name = "Dataset/" + data_sets[ds][0]
    tst_file_name = "Dataset/" + data_sets[ds][1]

    trn_words, trn_tags, tst_words, tst_tags, words_ind, tags_ind =
load_data(
        trn_file_name, tst_file_name
    )
```

برای هرکدام از دیتاست ها داده های ترین و تست را میخوانیم با استفاده از تابع load_data

```
tag_sharp = np.where(tags_ind == "###")[0][0]
word_sharp = np.where(words_ind == "###")[0][0]
```

کدی که برای tag_sharp استفاده شده هم تو tagsها و هم تو کلامات را بدست میاریم تست

```
n_tags = tags_ind.shape[0]
n_words = words_ind.shape[0]
```

تعداد کلمات و تگها رو بدست میاریم

```
a, b, p = add_one_smoothing(trn_words, trn_tags, tag_sharp, n_tags, n_words)
```

از add_one_smppthig ورودیها را صدا میزنیم برامون a,b,p را پیدا کند.

```
prd_tags = apply_viterbi(a, b, p, tst_words, tags_ind, words_ind, word_sharp)
```

در نهایت از apply_viterbi استفاده میکنیم که تگهایی که پیشبینی شدند را برگرداند.

```
tst_actual_tags = [tags_ind[i] for i in tst_tags]
tst_actual_words = [words_ind[i] for i in tst_words]
```

تگهای واقعی هم که کدگذاری شده بودند را از کدگذاری خارج کند برای داده های تست.

```
s = sum([prd_tags[i] == tst_actual_tags[i] for i in range(len(prd_tags))])
acc = s / len(prd_tags)
print("{} accuracy is {}".format(ds, acc))
```

اینجا دقت را حساب میکنم و با مقایسه کردن اینها دقت بدست میاد و در نهایت دقت را در خروجی چاپ کند.

```
f = open("{}_add_one_smoothing.txt".format(ds), "w")
for word, tag, prd in zip(tst_actual_words, tst_actual_tags, prd_tags):
    f.writelines(word + "/" + tag + "/" + prd + "\n")
f.close()
```

و همجنین درون یک فایل به اسم خود دیتاست و اضافه شدن add_one_smoothing.txt به انتهای آن یک ستون ایجاد کنم که در هر سطر با 2 تا // از هم جدا شده خود کلمه و تگ واقعی اون کلمه و تگی که براش پیشبینی شده را در فایل مینویسم.

```
def forward_backward(a, b, p, trn_seqs, v, q, forward_fcn, backward_fcn,
max_iter=10):
```

الگوریتم forward_backward

trn_sequence لیستی از جملات است یعنی جملات از داخل فایل خونده شدن و تبدیل شدن به لیستی از جملات

v تعداد کلمات را نشان میدهد.

q تعداد تگها حالت ها را نشان میدهد.

تابع forward_fcn, backward_fcn هم اینجا به صورت ورودی دریافت میکنیم.

max_itr تعداد حداکثر تکرار

```
alpha = np.exp(forward_fcn(a, b, p, o))
beta = np.exp(backward_fcn(a, b, o))
```

forward , backward را صدا میزنیم تا alph , beta را حساب کند.

```
N, T = alpha.shape
```

N,T خواهد شد shap alpha چون دیگه چون ابتدا سطرهای آلفا داره نشون میده چندتا حالت داریم و T هم نشان دهنده تعداد کلمات داخل جملمون است.

```
s = alpha.sum(axis=0)
alpha = alpha / s[np.newaxis, :]

s = beta.sum(axis=0)
s[s == 0] = 1
beta = beta / s[np.newaxis, :]
```

همون چیزی که گفته شده اینجا alpha را نرمالسازی میکنیم و بتا هم نرمالسازی میکنیم.

Beta قبل از اینکه از تابع exp استفاده کنیم داخلش مقدار صفر هم ایجاد میشد بنابراین من مقدار صفر را با حاصل جمع جایگذین میکردم کردم تا وقتی تقسیم انجام میدیم مقدار Nan برامون ایجاد نشه. این خط میتونه حذف بشه چون داریم از exp استفاده میکنیم.

```
denum = alpha[:, -1].sum()
gamma = alpha * beta / denum
```

اینجا مخرج را حساب میکنیم مخرج کسری که داشتیم برای پیدا کردن pc

```
xee[i, j, t] = (
    alpha[i, t]
    * a[i, j]
    * b[j, o[t + 1]]
    * beta[j, t + 1]
    / denum
```

چون مخرج یکسان بود همیشه من یکبار اینجا حساب میکنم و در ادامه ازش استفاده میکنم.

برای gamma آلفا صرب در بتا درایه به درایه تقسیم به مخرج denum

برای حساب کردن زی xee دونه دونه

```
for i in range(N):
            for j in range(N):
                a[i, j] = xee[i, j, :].sum() / xee[i, :-1, :].sum()
        for j in range(b.shape[0]):
            for vk in range(b.shape[1]):
                b[j, vk] = gamma[j, o == vk].sum() / gamma[j, :].sum()

return a, b
```

برای حساب کردن اون یکی هم

```
a[i, j] = xee[i, j, :].sum() / xee[i, :-1, :].sum()
```

به همین صورت که a[i,j] برابر میشه با زی i و j و زمان را که داخلش سیگما ایجاد کرده که : به همین دلیل است و sum() که بگیریم میشه سیگمایی که همونجا ایجاد شده و برای بعدی تا t-1 رفته جلو که منم دقیقا همین کار رو کردم و همه را به هم جمع کردم.

```
b[j, vk] = gamma[j, o == vk].sum() / gamma[j, :].sum()
```

برای پیدا کردن b هم گفته اونایی رو در نظر بگیرید که o==vk برای تگ k ام که خروجی در اون حالت برابر با اون تگ باشه اینها را با هم جمع کرده و تقسیم بر همشون کنید تا نرمال بشه و در نهایت a,b را به عنوان خروجب برگردونه.

```
def load_raw(file_name, words_ind):
```

کاری که انجام میشه ابتدا اطلاعات از

```
df = pd.read_csv(file_name, header=None, sep=" ")
```

داخل فایلraw خونده میشه .

```
words = df.values
```

کلمات داخلش استخراج میشه.

```
ind = np.zeros(words.shape)
```

یک آرایه دیگه ایجاد میکنیم آرایه اندیس. با استفاده از تعداد کلمات

```
for word in words_ind:
    ind = np.logical_or(ind, words == word)
```

و در نهایت اندیس اون کلماتی را اینجا پیدا میکنم که داخل داده های تست وجود داشت و لی داخل داده های فایل raw هم وجود داشته و اون داده هایی که وجود نداشته را حذف میکنم.

```
words = words[ind]
coded_words = np.zeros((words.shape[0],), dtype=int)
for i, word in enumerate(words_ind):
    coded_words[words == word] = i
```

بعد طبق همون کدگذاری که داخل فایل ترین با back انجام دادیم اینجا هم همان کد گذاری را دقیقا انجام میدیم.

```
ind = np.where(words == "###")[0]
ind = np.insert(ind, 0, -1) + 1
seqs = []
for i in range(ind.shape[0] - 1):
    seqs.append(coded_words[ind[i] : ind[i + 1]])

return seqs
```

بعد با استفاده از اندیس کلمه 3 تا شارپ محدوده بین جملات را مشخص میکنم

جملات را از هم جدا میکنم و داخل یک لیست قرار میدم به عنوان خروجی استفاده میکنم.

```
seqs = load_raw(raw_file_name, words_ind)[:100]
```

داده های raw هم میخونیم.

```
a, b = forward_backward(
    a, b, p, seqs, tags_ind.shape[0], a.shape[0], forward_alg_1,
backward_alg_1
```

بعد forward, backward را صدا میزنیم. میتوانیم درون forward, backward الگوریتم 1 را صدا کنیم یا 2 را.

خروجی ها را مثل قبل چاپ می شود.