NLP100 BASIC Hands-on

Attention!

This lecture is made for beginners.
After this, you can easily tackle NLP100 CH6-9.
(If you can solve NLP100 alone, you don't need to take the lecture.)

Prepare

The main purpose of this lecture is to get accustomed with coding in ML.

Copy this Colab into your own drive to solve the exercises.

Section 1

- NLP 処理の流れ
- EDA with dataframe
- pre-process
- モデル選択
- 学習
- hyper parameter setting

NLP 処理の流れ

一般的なNLPでは、以下の流れで行います。

- EDA: Explanatory Data Analysis
- pre-process
- choose a model
- train&predict
- hyper parameter setting

今回はこれに従って、文章を分類するモデルを作ってみます。
では順に見ていきましょう。

EDA with dataframe

機械学習を行う際には、EDA(Explanatory Data Analysis)を行い、採用すべき特徴量やモデルを検討します。
pandas.dataframeはSQLチックに表データにアクセスするためのライブラリです。若干処理が遅いですが、デファクトスタンダードとなっています。

さっそくQ1を解きましょう
(dataframeに不慣れな方は、HINTを見てから解きましょう)

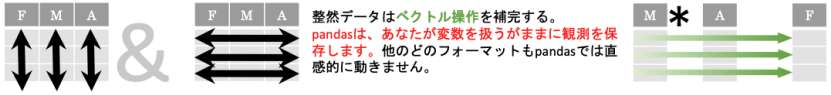
参考: 公式チートシート(jp,en)、データサイエンス100本ノック

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

整然データ（Tidy Data） – pandasにおける議論の基盤

整然データ においては:



各変数は自身の列に保存されます

各observation は自身の行に保存されます

整然データはベクトル操作を補完する。
pandasは、あなたが変数を扱うがままに観測を保存します。他のどのフォーマットもpandasでは直感的に動きません。

M * A

文法 – DataFrameの作成

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```
各column(列)の値をセットする

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
各row(行)の値をセットする

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
         names=['n','v']))
```
MultiIndexでDataframeを作成する

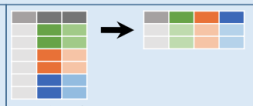
メソッドチェーン

pandasにおける多くのメソッドはDataframeを返します。そのためメソッドの返り値にそのままほかのメソッドを適用するメソッドチェーンが便利に使えます。この手法はコードの可動性を大いに向上します。

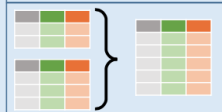
```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

データの整形(Reshaping Data) – データセットのレイアウト変更

pd.melt(df)
各column(列)をrow(行)へ.



df.pivot(columns='var', values='val')
各row(行)をcolumn(列)へ



pd.concat([df1,df2])
DataFrameのrow(行)を連結

pd.concat([df1,df2], axis=1)
DataFrameのcolumn(列)を連結

df.sort_values('mpg')
column(列)の値を使ってrow(行)をソート(昇順)

df.sort_values('mpg',ascending=False)
column(列)の値を使ってrow(行)をソート(降順)

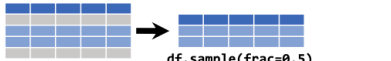
df.rename(columns = {'y':'year'})
DataFrameのcolumn(列)名を変更

df.sort_index()
DataFrameのindexを使ってソート

df.reset_index()
DataFrameのindexをリセット

df.drop(columns=['Length','Height'])
指定した長さのcolumn(列)を削除

Observations(行)の一部を抜き出し

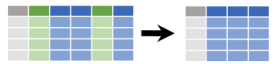


df[df.Length > 7]
与えられた条件に合った行を抜き出す
df.drop_duplicates()
値の重複するrow(行)を除外
df.head(n)
最初のn行を取得
df.tail(n)
最初のn行を取得

df.sample(frac=0.5)
fracに与えた割合でランダムで行を取得
※fracは割合(1 = 100%)
df.sample(n=10)
n行をランダムで取得
df.iloc[10:20]
指定位置の行を取得
df.nlargest(n, 'value')
'value'列のn行を降順で取得
df.nsmallest(n, 'value')
'value'列のn行を昇順で取得

Logic in Python (and pandas)			
<	より小さい	!=	等しくない
>	より大きい	df.column.isin(*values*)	valuesが含まれているcolumnに場合trueを返す
==	と等しい	pd.isnull(*obj*)	nullである
<=	以下	pd.notnull(*obj*)	nullでない
>=	以上	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

変数(列)からの一部取得



df[['width','length','species']]
複数column(列)を列名を指定して取得
df['width'] *or* **df.width**
1つのcolumn(列)を列名を指定して取得
df.filter(regex='*regex*')
column(列)を正規表現でフィルタリング

regex (正規表現) の例	
'\.'	ピリオド'.'を含む文字列にマッチ
'Length$'	末尾に'Length'のある文字列にマッチ
'^Sepal'	冒頭に'Sepal'のある文字列にマッチ
'^x[1-5]$'	'x'で始まり且つ末尾が'1~5のいずれかである文字列にマッチ
'^(?!Species$).*'	'Species'以外の文字列とマッチ

df.loc[:,'x2':'x4']
x2からx4までの全てのcolumn(列)を取得
df.iloc[:,[1,2,5]]
1,2,5番目(indexが5番目)の列を取得(indexは0から数える)
df.loc[df['a'] > 10, ['a','c']]
与えられた条件に合ったrow(行)で且つ指定されたcolumn(列)を取得

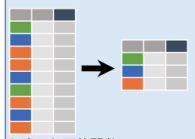
データの要約

df['w'].value_counts()
変数の出現回数をカウント
len(df)
DataFrameの行数を出力
df['w'].nunique()
ユニークな値をカウントして出力
df.describe()
Basic descriptive statistics for each column (or GroupBy)

pandasは様々な種類のpandasオブジェクト(DataFrame columns, Series, GroupBy, Expanding and Rolling(下記参照))を操作する**summary functions(要約関数)**を提供し、各グループに対して1つの値を返します。DataFrameに適用された場合、結果は各column(列)にSeries型で返されます。例:

sum()
各オブジェクトの値を合計
count()
各オブジェクトのNA/null以外の値をカウント
median()
各オブジェクトの中央値を取得
quantile([0.25,0.75])
各オブジェクトの分位値を取得
apply(*function*)
各オブジェクトにを適用
min()
各オブジェクトの最小値を取得
max()
各オブジェクトの最大値を取得
mean()
各オブジェクトの平均を取得
var()
各オブジェクトの分散値を取得
std()
各オブジェクトの標準偏差を取得

データのグループ化



df.groupby(by="col")
"col"列の値でグループ化したGroupByオブジェクトを返す
df.groupby(level="ind")
インデックスレベル"ind"でグループ化したGroupByオブジェクトを返す

上述した要約関数(summary function)は全てgroupにも適用可能です。その他GroupByの関数:

size()
各グループの長さ
agg(*function*)
関数を使ってグループを集計

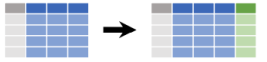
下記関数もgroupに対して適用できます。この場合、関数はグループ毎に適用され、返されるベクトルの長さは元のDataFrameと同じになります。

shift(1)
1行ずつ後ろにずらした値をコピー
rank(method='dense')
ランク付け(同数はギャップなしで計算)
rank(method='min')
ランク付け(同数は小さい値にする)
rank(pct=True)
[0~1]の値でランク付け
rank(method='first')
ランク付け。同数の場合indexが小さい方が上位
shift(-1)
1行ずつ前にずらした値をコピー
cumsum()
累積和
cummax()
累積最大値
cummin()
累積最小値
cumprod()
累積積
min(axis=1)
要素ごとの最大値を取得
abs()
絶対値を取得

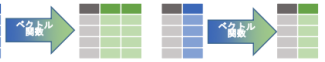
欠損データを扱う

df.dropna()
NA/nullを含むrow(行)を除外する
df.fillna(value)
NA/nullをvalueに置換

新しいColumn(列)の作成



df.assign(Area=lambda df: df.Length*df.Height)
1つ以上の新たなcolumn(列)を計算して追加
df['Volume'] = df.Length*df.Height*df.Depth
新たなcolumn(列)を1つ追加
pd.qcut(df.col, n, labels=False)
column(列)の値をn分割



pandasはDataFrameの全てのcolumn(列)または選択された1列(Series型)を操作できる**vector functions(ベクトル関数)**を提供します。それらの関数は各列(column)に対してベクトル値を返します。また、各Seriesには1つのSeriesを返します。例:

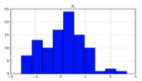
max(axis=1)
要素ごとの最大値を取得
clip(lower=-10,upper=10)
下限を-10,上限を10に設定してトリミング

window関数

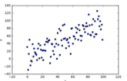
df.expanding()
要約関数を累積的に適用可能にした Expanding オブジェクトを返す
df.rolling(n)
長さnのwindowに要約関数を適用可能にしたRollingオブジェクトを返す

プロット(描画)

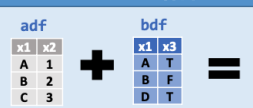
df.plot.hist()
各列のヒストグラムを描画



df.plot.scatter(x='w',y='h')
散布図を描画

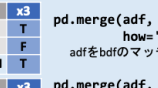


データの結合

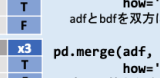


標準的な結合

pd.merge(adf, bdf, how='left', on='x1')
bdfをadfのマッチする行へ結合



pd.merge(adf, bdf, how='right', on='x1')
adfをbdfのマッチする行へ結合



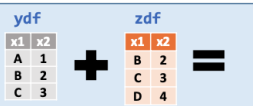
pd.merge(adf, bdf, how='inner', on='x1')
adfとbdfを双方にある行のみ残して結合

pd.merge(adf, bdf, how='outer', on='x1')
全ての値と行を残して結合

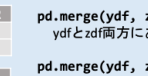
フィルタリング結合

adf[adf.x1.isin(bdf.x1)]
adfの中でbdfにマッチする行

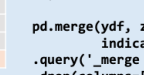
adf[~adf.x1.isin(bdf.x1)]
adfの中でbdfにマッチしない行



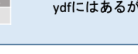
集合ライクな結合



pd.merge(ydf, zdf)
ydfとzdf両方にある行



pd.merge(ydf, zdf, how='outer')
ydfとzdfの両方もしくは片方にある行




```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
ydfにはあるがzdfにはない行

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

pre-process

生のテキストではうまく学習できないため、前処理が必要です。

- テキストのクリーニング: 目標のテキスト以外を削除する
- 単語分割(token): 形態素解析を行い、単語単位にばらす
- 単語の正規化: 文字種や表記ゆれを統一する、数字を0に置き換える
- ストップワードの除去: 機能語(助詞や助動詞)を除去する
- 単語のID化: Bag of WordsまたはNNでベクトル化
- バディング: 系列長を揃える(NN学習用)

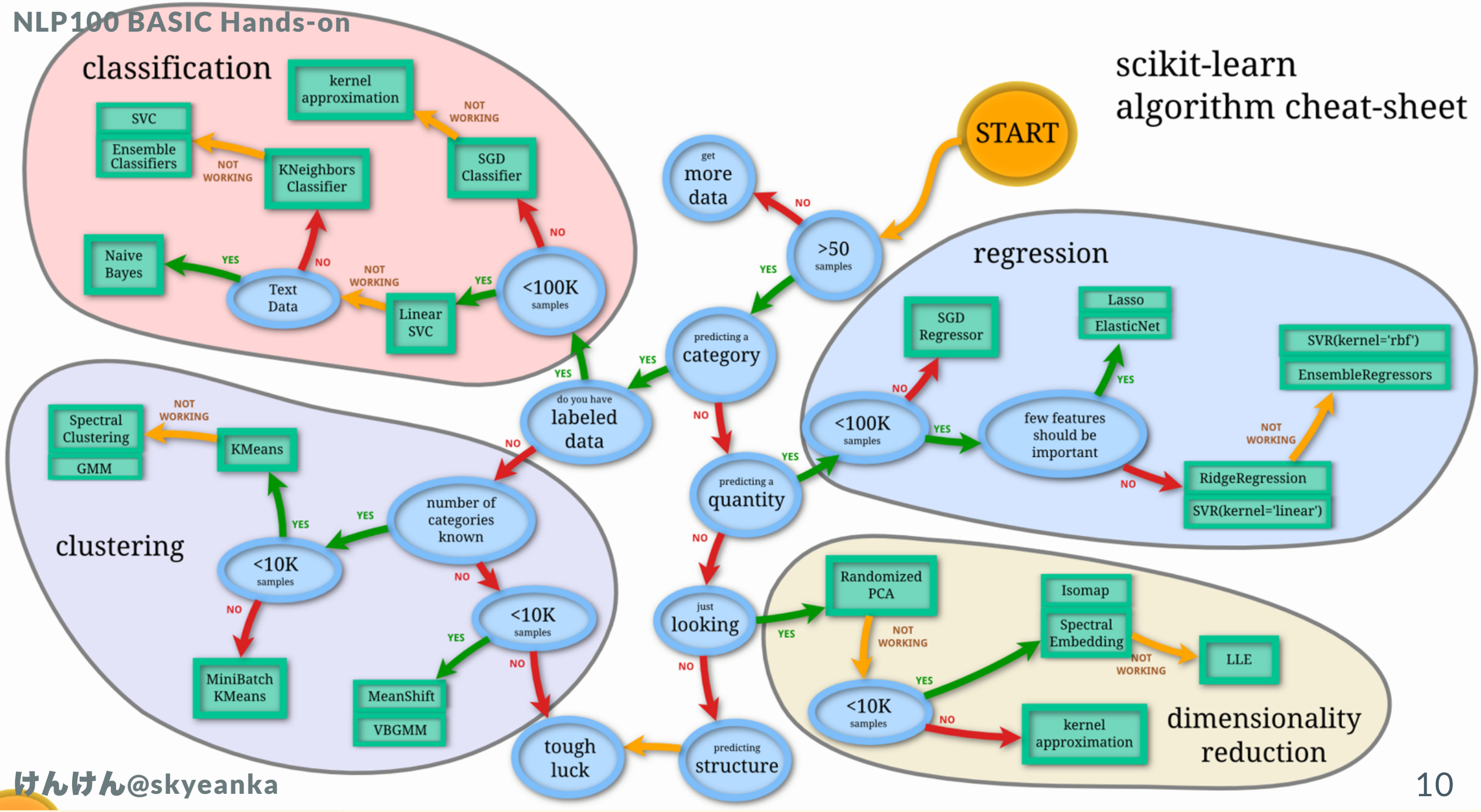
Q2で前処理を練習しましょう。

モデル選択

モデル選択の際は、とりあえずのモデル(baseline)を作り、そこからベースモデルを基準に様々なモデルを選定していくことになります。

機械学習のモデルは、たいていsklearnに実装されています。

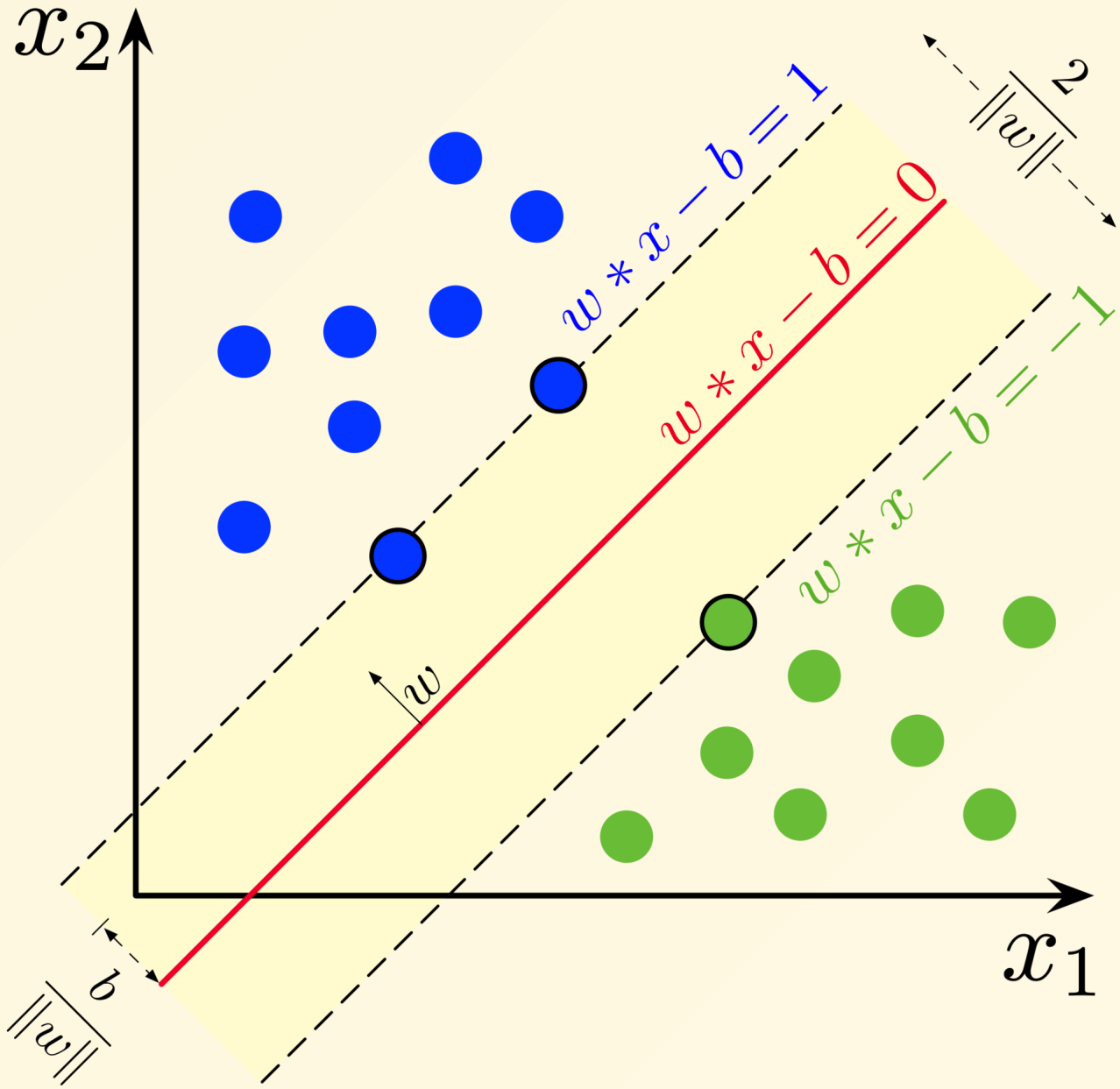
今回は多クラス分類問題のモデルを作ります。次ページのチートシートに従い、LinearSVC(線形SVM)を使うことにしましょう。


scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
NO
YES
predicting a category
NO
YES
do you have labeled data
NO
YES
predicting a quantity
NO
YES
just looking
NO
YES
predicting structure
tough luck
<100K samples
NO
YES
Linear SVC
NOT WORKING
Text Data
NO
YES
Naive Bayes
KNeighbors Classifier
NOT WORKING
SGD Classifier
NO
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
classification
<100K samples
NO
YES
SGD Regressor
few features should be important
NO
YES
Lasso
ElasticNet
RidgeRegression
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
SVR(kernel='linear')
regression
number of categories known
NO
YES
<10K samples
NO
YES
KMeans
Spectral Clustering
NOT WORKING
GMM
<10K samples
NO
YES
MiniBatch KMeans
MeanShift
VBGMM
clustering
Randomized PCA
NOT WORKING
<10K samples
NO
YES
kernel approximation
Isomap
Spectral Embedding
NOT WORKING
LLE
dimensionality reduction

学習

MLにおける学習では、目的関数を立て、それが最小(最大)となるように行います。
今回使う線形SVMでは、マージン $||w||$ が最大となるように目的関数が立てられます。

では、実際に学習してみましょう(Q3)



hyper parameter setting

先程のA3では、LinearSVCをパラメータを指定せず、初期値のまま学習しましたが、例えば損失関数をL1とすると、スコアが変動します。

このような学習前に指定する必要があるパラメータをハイパーパラメータと呼び、trainデータで学習し、validationデータを利用し評価します。実際のデータに対しての評価も行いため、ここではtestデータは使用しません。


今日はここまで

Section 2

- Word2Vec
- Neural Network
- NNの学習
- CNN
- RNN
- LSTM
- Attention
- Transformer
- 時系列NNまとめ
- Huggingface🤗

Word2Vec

Word2Vecは、後述するNNを使って、単語を**計算可能な**分散(ベクトル)表現へ変換する手法です。

```
w2v['父']-w2v['男']+w2v['女'] == w2v['母']
```

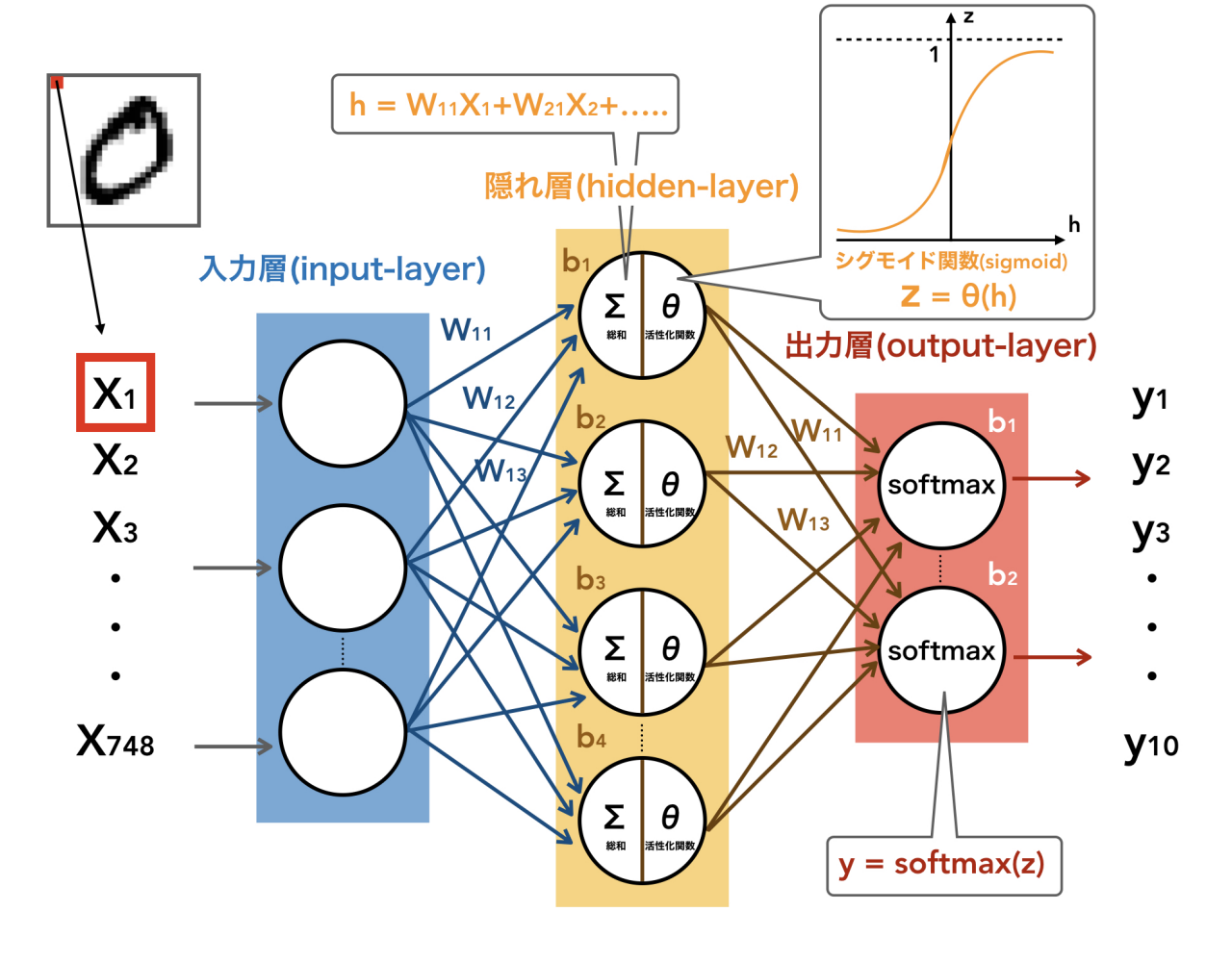
Q2で作った単語ベクトルを、Word2Vecで作り直してみましょう(Q4)

https://towardsdatascience.com/understanding-

Neural Network

脳の神経回路を模した数理モデル
隠れ層を増やすことで精度の高い非線形な判別が可能となります

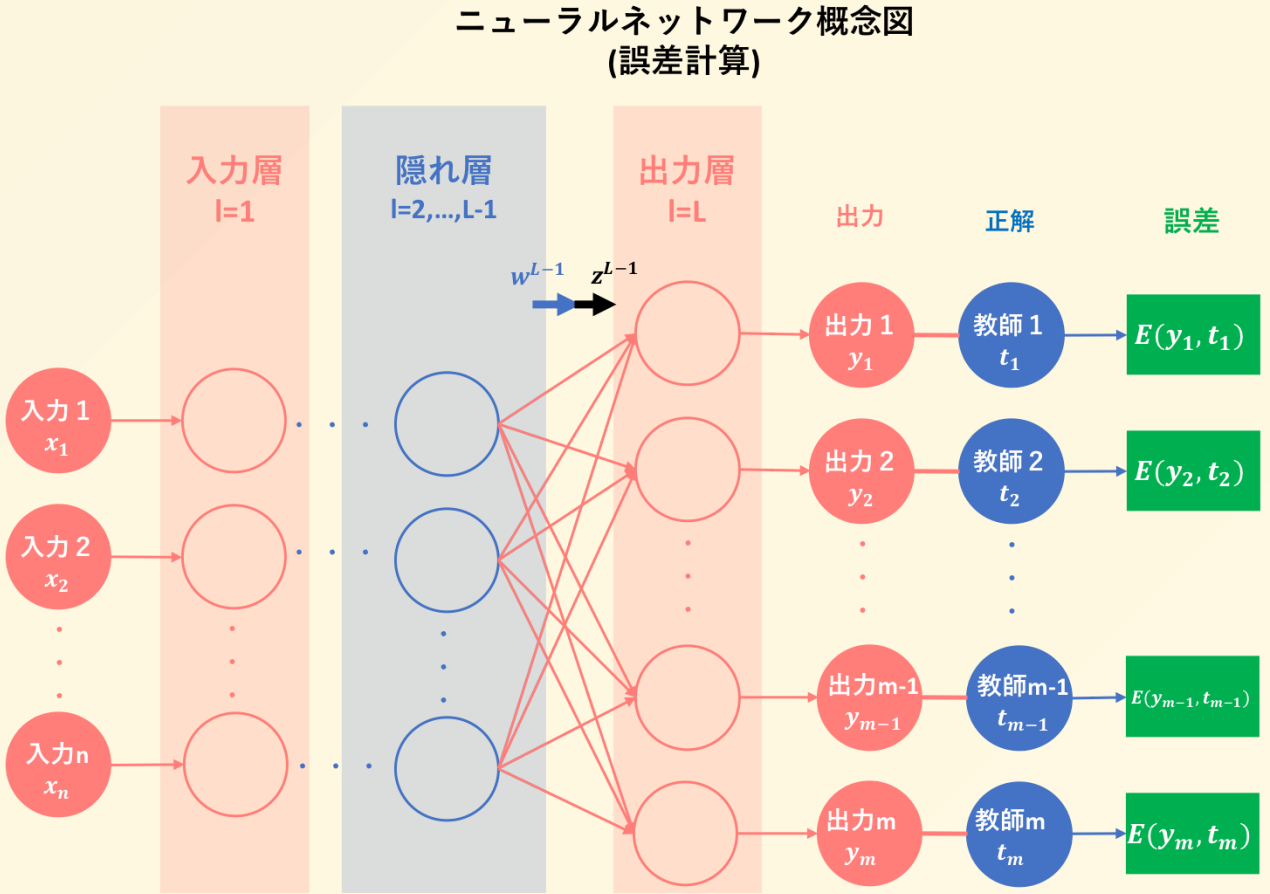
重みwを上手に学習(自動的に計算)できるようになり、今日機械学習の主役となっています。



NNの学習

重みの学習では、出力と教師データとの差分が小さくなるように行います。
具体的には、損失関数を立て、勾配降下法などで重みを探索することになります。

Q5でシンプルなNNを実装してみましょう。

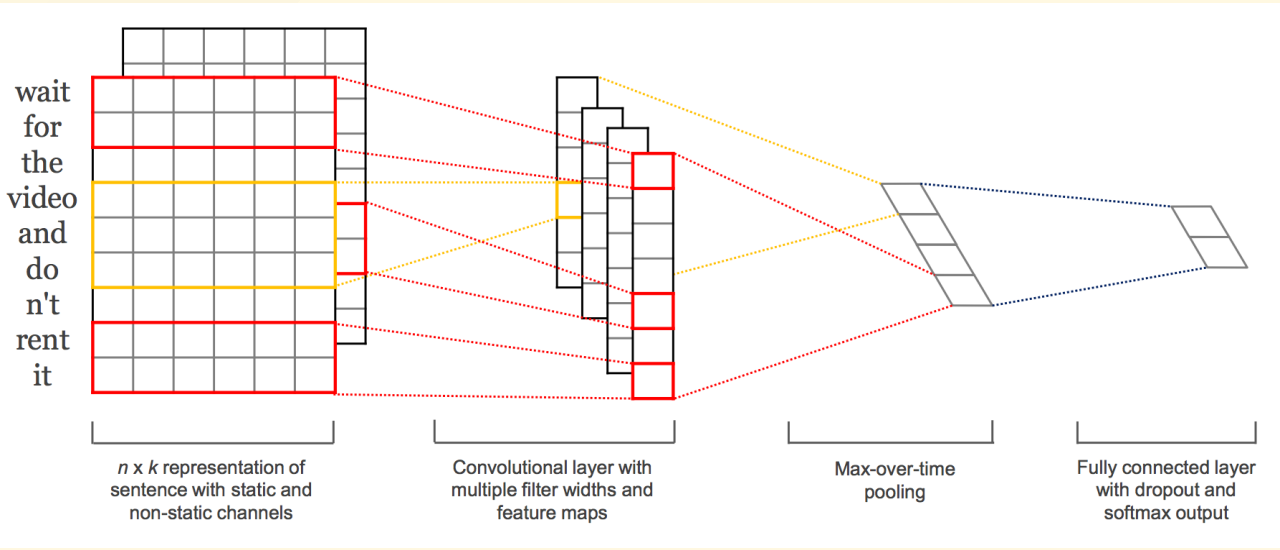


https://qiita.com/sakigakeman/items/44af9ffde055

CNN

CNNは画像処理で有名ですが、NLPでも有効です。

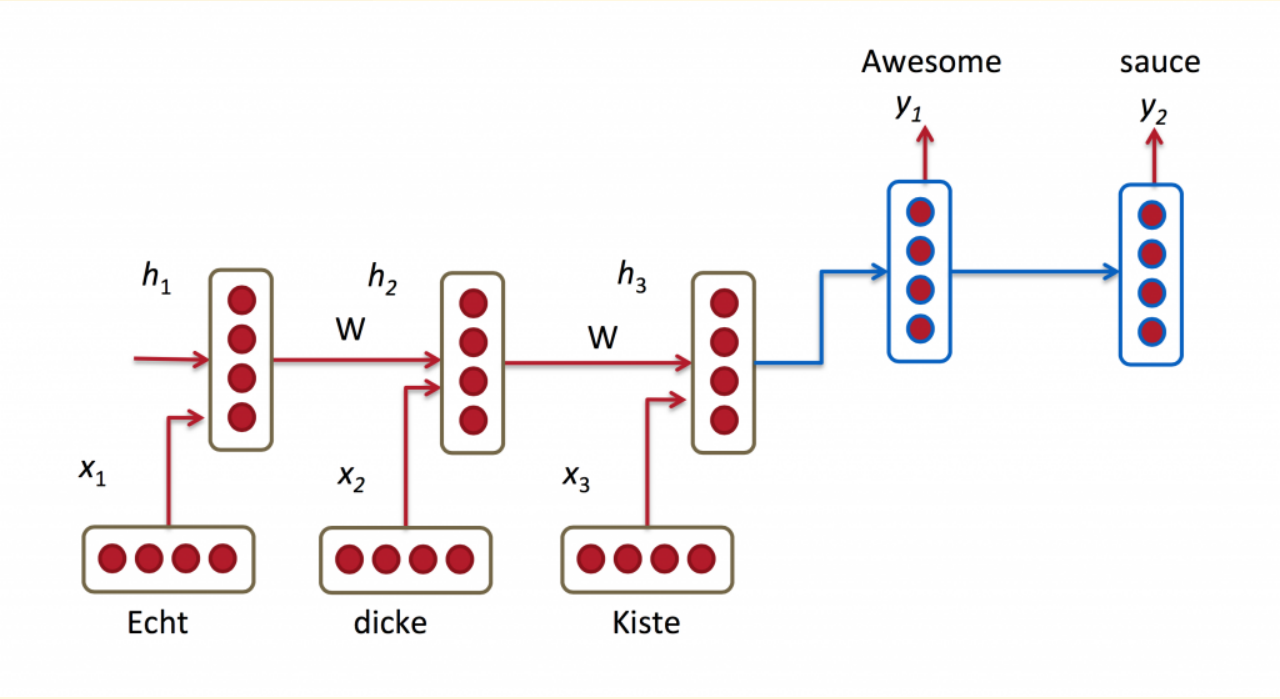
例えば、文章と単語ベクトルで構成した行列でCNNを行うと良いパフォーマンスが得られます Kim(2014) 解説記事(jp)



RNN

時系列データを扱えるようにしたNNです。
隠れベクトルhをinputに加えることで、前回までの入力が反映されています。

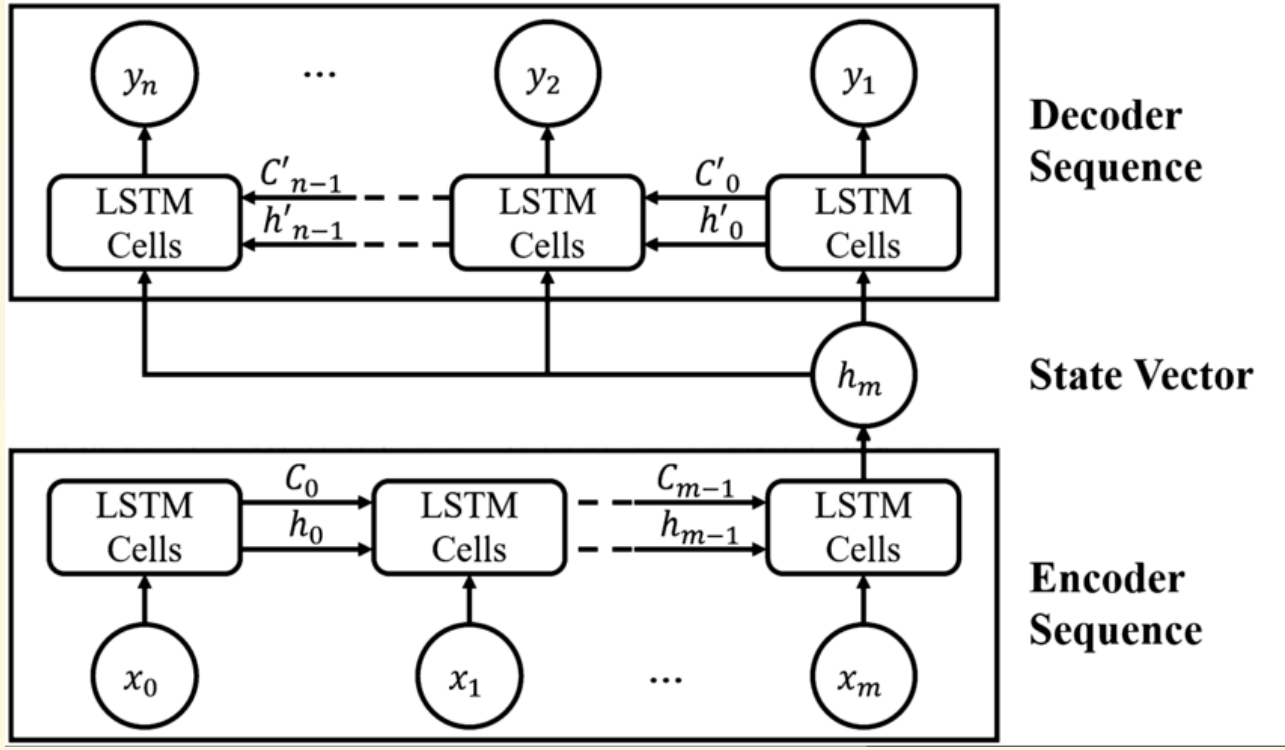
シンプルですが、時系列データが長くなると、隠れベクトルにうまく埋め込みできません。



LSTM

過去の入力を短期記憶hと長期記憶cに分けて学習するモデルです。

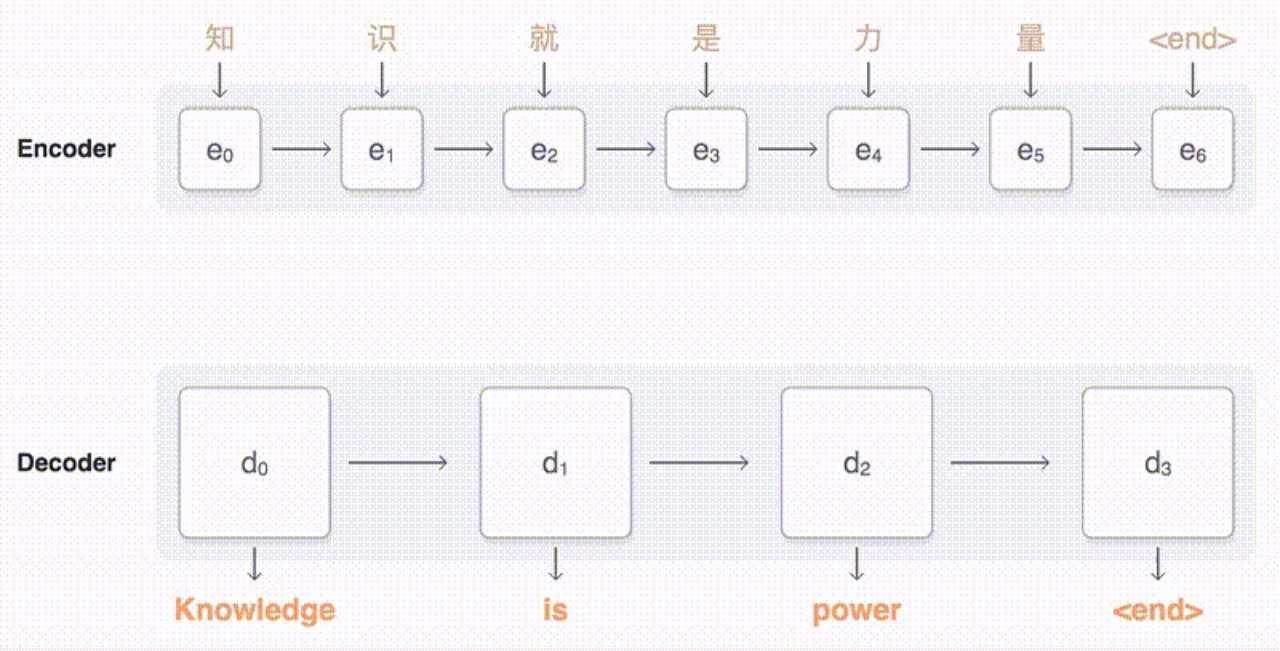
計算量は増えますが、RNNよりも過去の文脈を捉えられるようになりました。



Attention

どの入力に注目すれば良い出力を得られるか自動計算できるようにした仕組みです。

Encoder/DecoderにLSTMを使ったモデルの改良という形で提案されました。

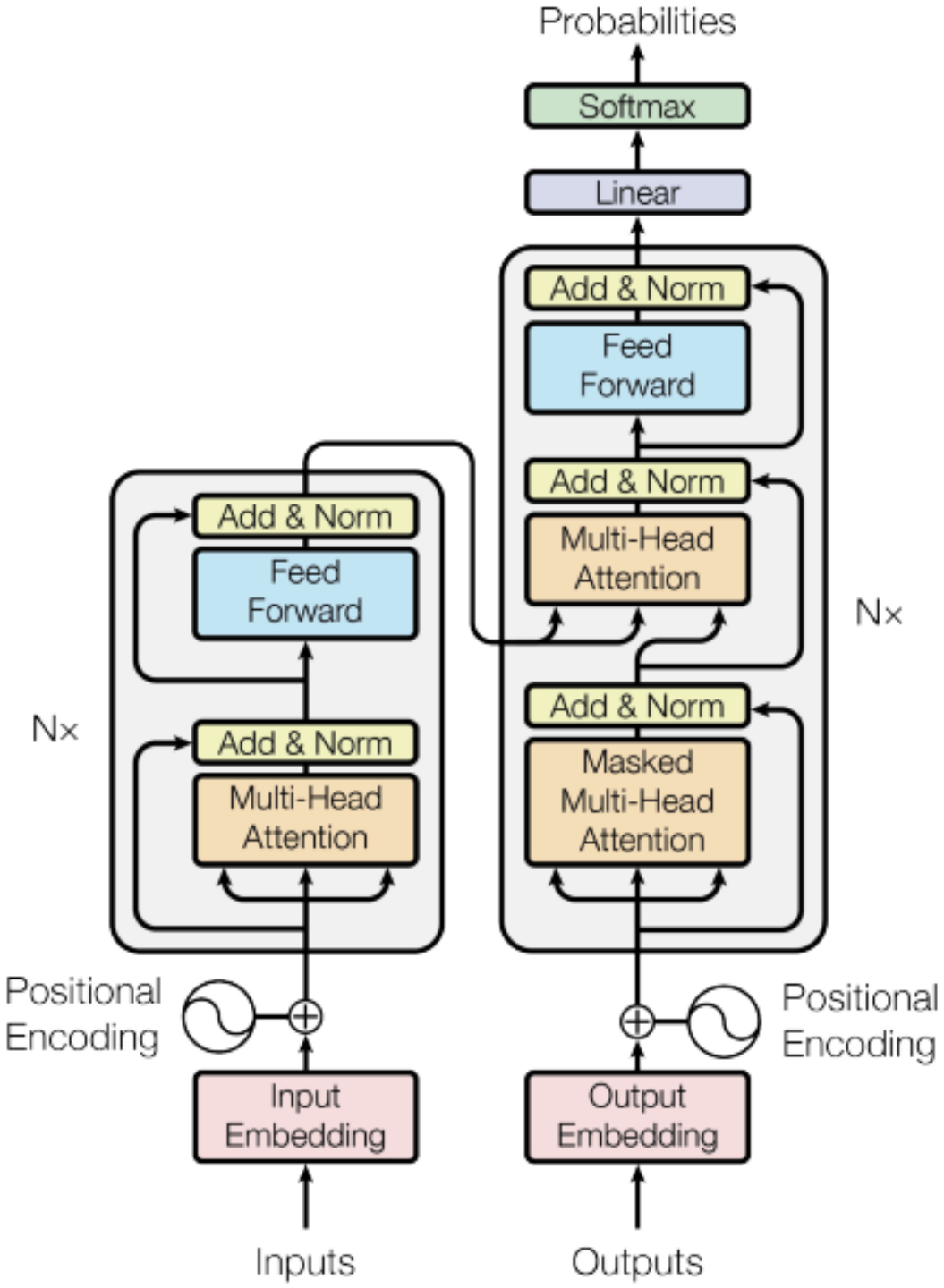


Transformer

2017年登場以来、自然言語処理の基本モデルとなっています。

系列変換(Encoder・Decoderモデル)に、文章の各単語ごとの関連・類似度の評価(Attention)を組み込むことで、より長い文脈を学習できるようになりました。

Vaswani, Ashish et al.(2017)



時系列NNまとめ

- RNN: 過去入力を加味した出力ができるようになったNNモデル
- LSTM: RNNより長い過去入力を加味できるようになったNNモデル
- Attention:
  アライメント(出力する際にどの入力に着目すべきか)を自動化した仕組み。Encode/Decode自体はRNNやLSTMなどで行う。
- Transformer:
  Encode/DecodeごとAttentionでできるようにしたNNモデル、
  計算効率が向上したことで表現力も大きく向上

Huggingface🤗

Huggingface🤗はMLでポピュラーなライブラリで、訓練済みのモデルやデータセットを簡単に利用できます。
大規模な言語モデルを訓練するには莫大なコストが必要ですので、Huggingfaceのような訓練済みのモデルを利用することが一般的です。

ここでは、DeBERTaV3というモデルを使って、Q3,Q5で解いた分類問題を解いてみましょう(Q6)

参考: Huggingface Tutorial

EoF