

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

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、INFO FOR YOU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>

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'])
各行(行)の値をセットする
```

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

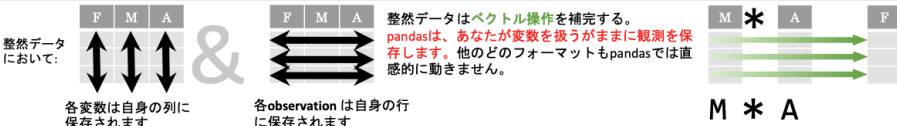
```
df = pd.DataFrame(
    {"a": [4, 5, 6],
     "b": [7, 8, 9],
     "c": [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2), ('e', 2)],
        names=['n', 'v'])
MultiindexでDataframeを作成する
```

メソッドチェーン

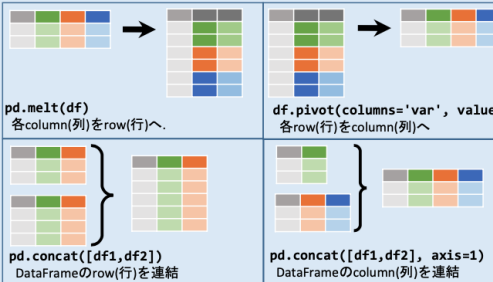
pandasにおける多くのメソッドはDataFrameを返す。そのためにメソッドの呼び出しは、そのメソッドの戻り値に続けてメソッドを呼び出すことができる。この手法はコードの可読性を向上させる。

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

整然データ (Tidy Data) - pandasにおける議論の基礎



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



```
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.sample(frac=0.5)
df.sample(n=10)
df.iloc[10:20]
df.nlargest(n, 'value')
df.nsmallest(n, 'value')
df.head(n)
df.tail(n)
df.index
df.columns
```

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
'Lengths'	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'5'のいずれかである文字列にマッチ
'*(?!Species\$).*	Species以外の文字列にマッチ
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番目のindex(行)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)
各オブジェクトに適用
```

データのグルーピング

```
df.groupby(by='col')
'col'列の値でグルーピングしたGroupByオブジェクトを返す

df.groupby(level='ind')
インデックスレベル'ind'でグルーピングしたGroupByオブジェクトを返す

df.groupby().size()
各グループの長さ
```

agg(function)
関数を使ってグループを集計

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
```

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に設定してリミット
```

```
min(axis=1)
要素ごとの最小値を取得

abs()
絶対値を取得

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(同数は最小値にする)

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
```

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



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

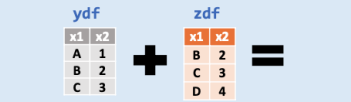
pd.merge(adf, bdf, how='right', on='x1')
adfをbdfのマッチする行へ結合

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
```

```
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
```



```
pd.merge(ydf, zdf)
ydfとzdfの両方にある行

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
```

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
- バディンク: 系列長を揃える(NN学習用)

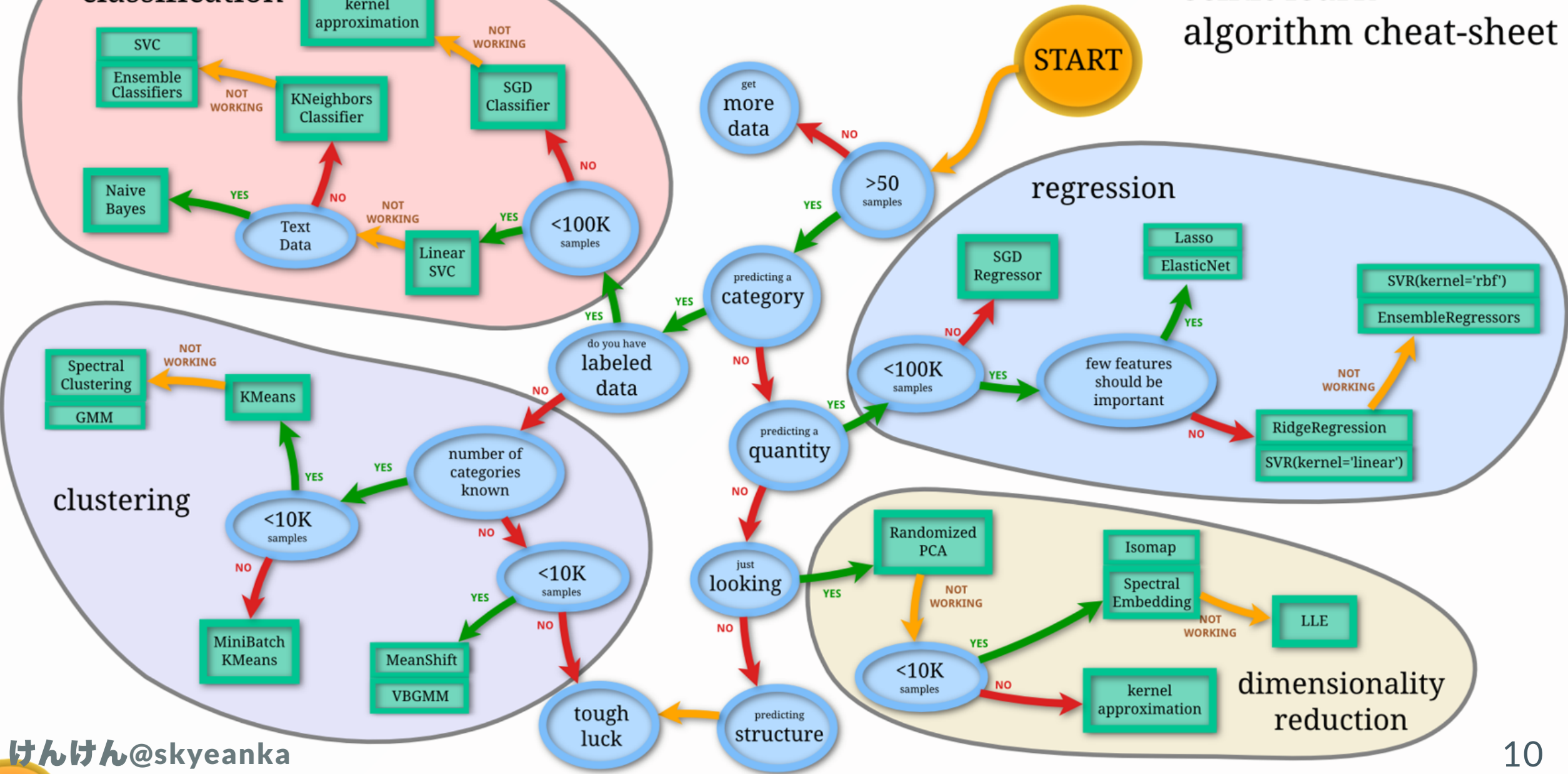
Q2で前処理を練習しましょう。

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。

今回の問題は多クラス分類問題ですので、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。

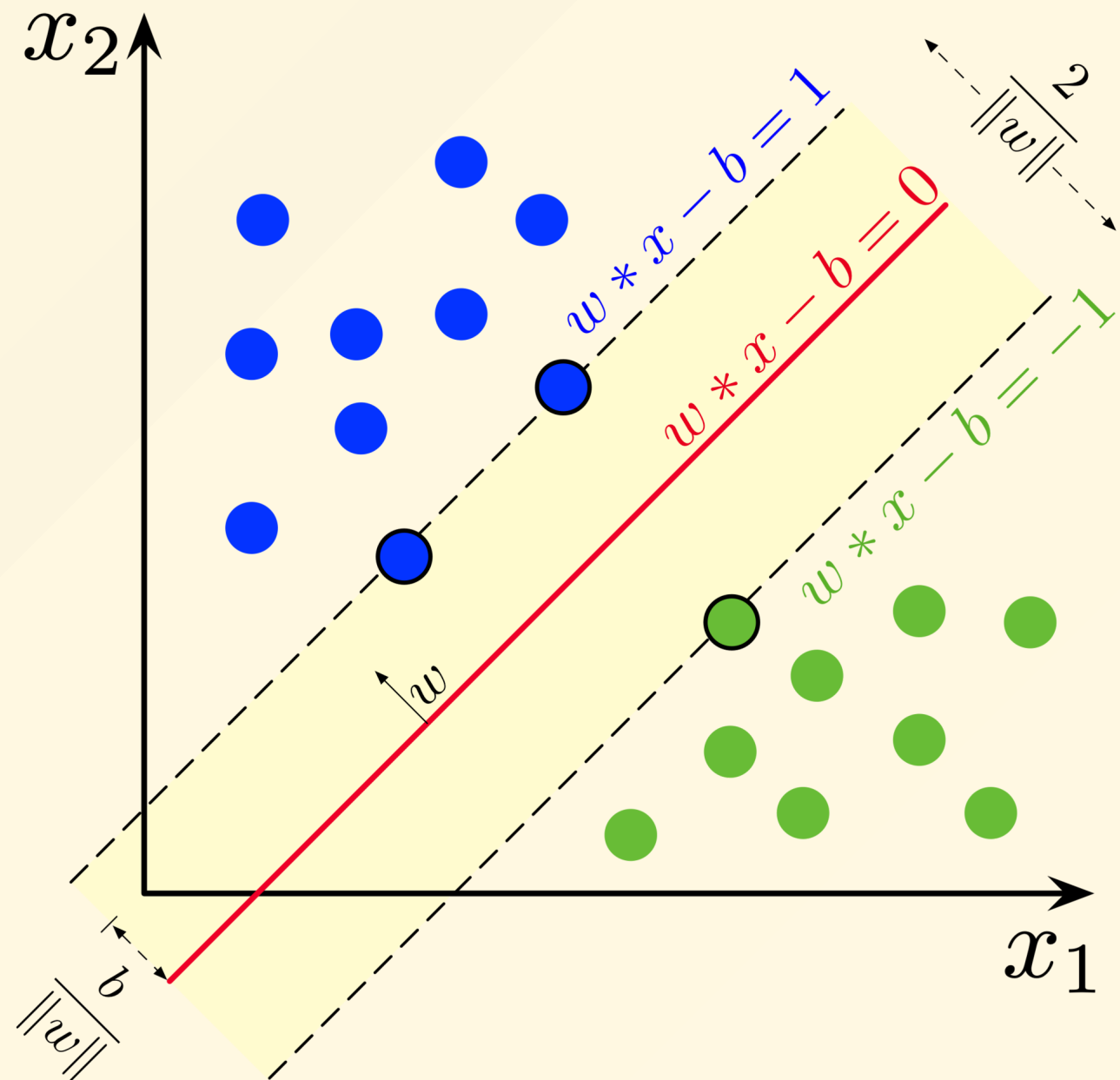


学習

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ため、ここではtestデータは使用しません。



Section 2

- [Word2Vec](#)
- [Neural Network](#)
- [RNN](#)
- [CNN](#)
- [Transformer](#)
- [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。

$$w2v[\text{'父'}] - w2v[\text{'男'}] + w2v[\text{'女'}] == w2v[\text{'母'}]$$

Q2で作った単語ベクトルを、Word2Vecで作り直してみましよう(Q4)

<https://towardsdatascience.com/understanding->

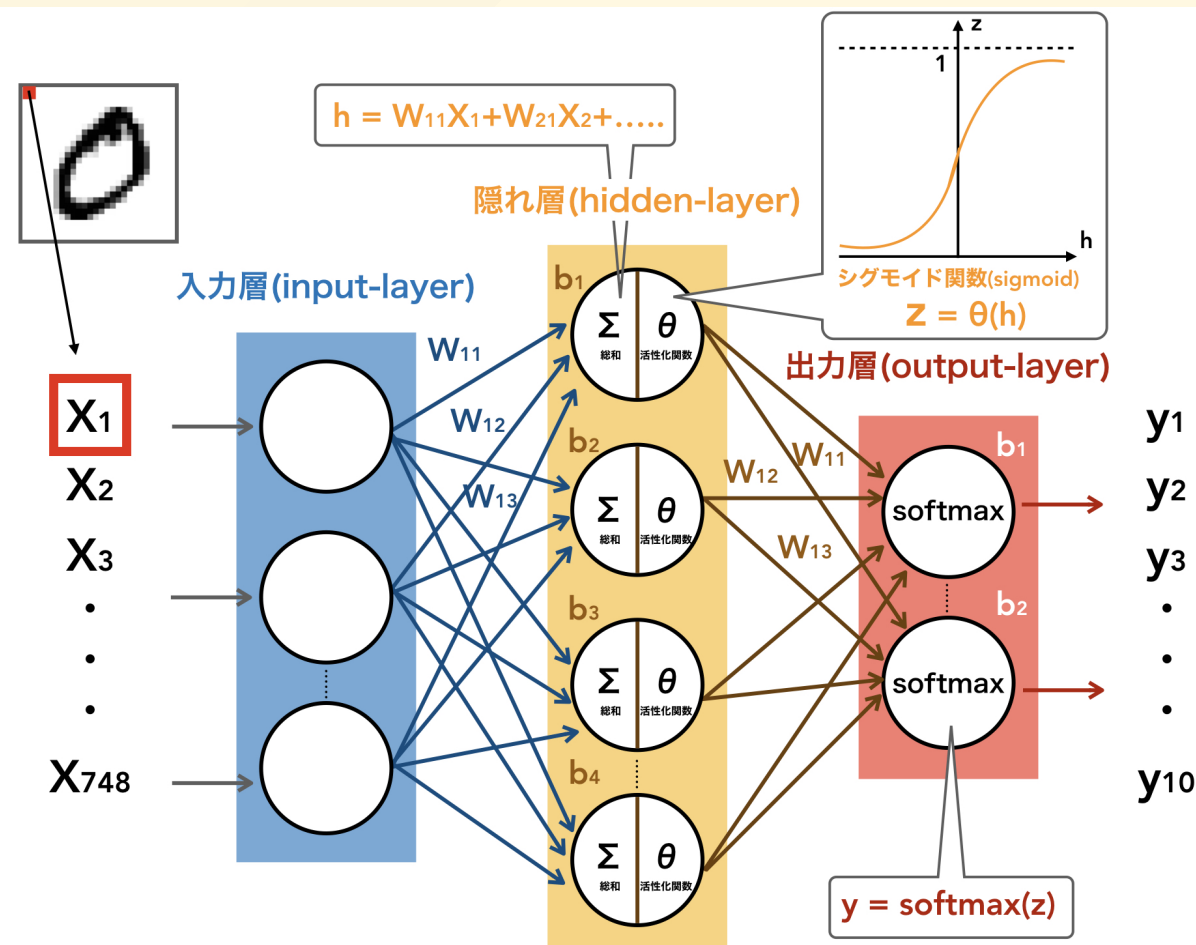


Neural Network

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

Q5でシンプルなNNを実装してみましょう

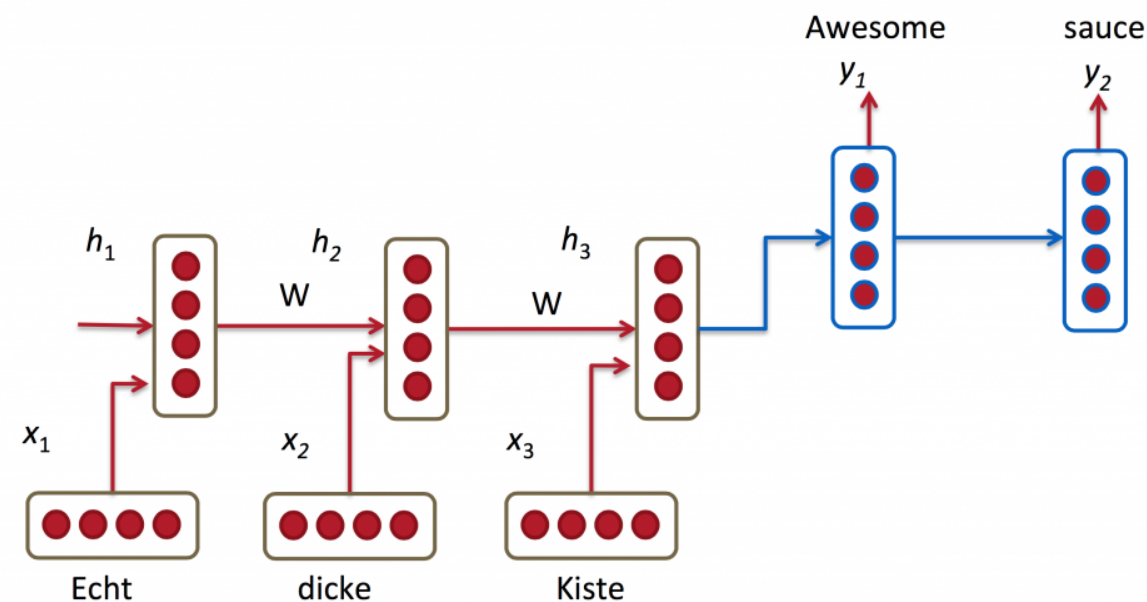


RNN

時系列データを扱えるようにしたNNです。

右図の通り、隠れ層 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。

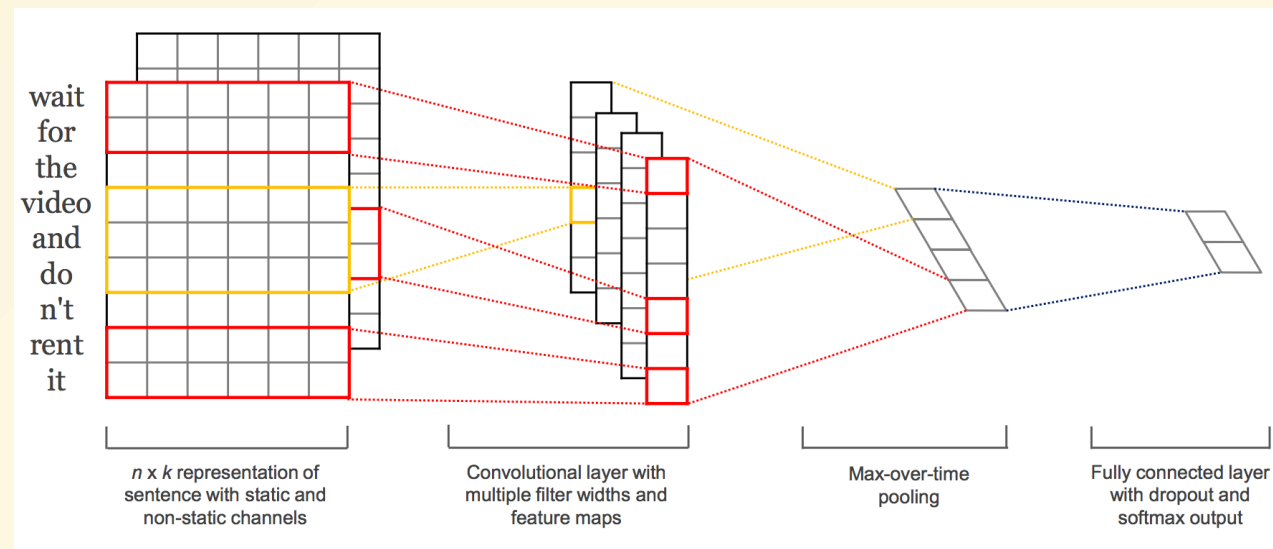
発展形にLSTMがあります。



CNN

CNNは画像処理で有名ですが、NLPでも有効です。

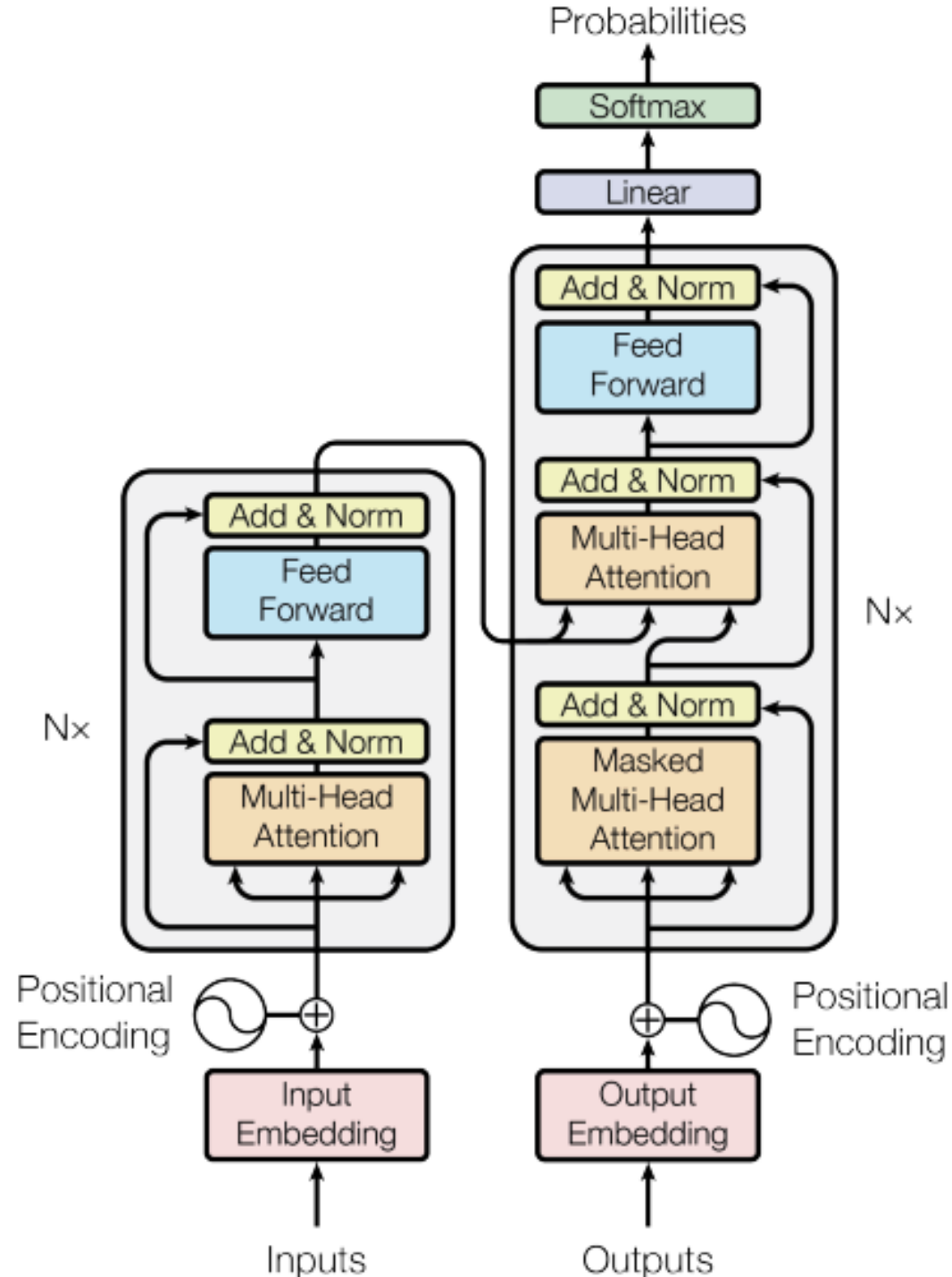
例えば、文章と単語ベクトルで構成した行列でCNNを行うと良いパフォーマンスが得られます [Kim\(2014\). 解説記事\(jp\).](#)



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。



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