napkinXC

Oct 17, 2022

Contents:

Note: Documentation is currently a work in progress!

napkinXC is an extremely simple and fast library for extreme multi-class and multi-label classification that implements the following methods both in Python and C++:

- Probabilistic Label Trees (PLTs) - for multi-label log-time training and prediction,
- Hierarchical softmax (HSM) - for multi-class log-time training and prediction,
- Binary Relevance (BR) - multi-label baseline,
- One Versus Rest (OVR) - multi-class baseline.

All the methods decompose multi-class and multi-label into the set of binary learning problems.

Right now, the detailed descirption of methods and their parameters can be found in this paper: Probabilistic Label Trees for Extreme Multi-label Classification

CHAPTER 1

Python Quick Start

1.1 Installation

Python (3.5+) version of napkinXC can be easily installed from PyPy repository on Linux and MacOS (Windows is currently not supported). It requires modern C++17 compiler, CMake and Git installed:

```
pip install napkinxc
```

or directly from the GitHub repository:

```
pip install pip install git+https://github.com/mwydmuch/napkinXC.git
```

1.2 Usage

napkinxc module contains three submodules: models that contains all the model classes and two additional modules

Minimal example of usage:

```
from napkinxc.datasets import load_dataset
from napkinxc.models import PLT
from napkinxc.measures import precision_at_k

X_train, Y_train = load_dataset("eurlex-4k", "train")
X_test, Y_test = load_dataset("eurlex-4k", "test")
plt = PLT("eurlex-model")
plt.fit(X_train, Y_train)
Y_pred = plt.predict(X_test, top_k=1)
print(precision_at_k(Y_test, Y_pred, k=1))
```

CHAPTER 2

Using C++ executable

napkinXC can also be built and used as an executable that can be used to train and evaluate models and make a prediction.

2.1 Building

To build napkinXC, first clone the project repository and run the following commands in the root directory of the project. It requires modern C++17 compiler, CMake and Git installed. Set CXX and CC environmental variables before running `cmake` command if you want to build with the specific C++ compiler.

```
cmake .
make
```

`-B` options can be passed to CMake command to specify other build directory. After successful compilation, `nxc` executable should appear in the root or specified build directory.

2.2 LIBSVM data format

napkinXC supports multi-label svmlight/libsvm like-format (less strict) and format of datasets from The Extreme Classification Repository, which has an additional header line with a number of data points, features, and labels.

The format is text-based. Each line contains an instance and is ended by a `\n` character.

```
<label>,<label>,... <feature>(:<value>) <feature>(:<value>) ...
```

`<label>` and `<feature>` are indexes that should be positive integers. Unlike to normal svmlight/libsvm format, labels and features do not have to be sorted in ascending order. The `:<value>` can be omitted after `<feature>`, to assume value = 1.

2.3 Usage

nxc executable needs command, i.e. train, test, predict as a first argument. -i/--input and -o/--output arguments needs to be always provided.

```
nxc <command> -i <path to dataset> -o <path to model directory> <args> ...
```

2.4 Command line options

```
Usage: nxc <command> <args>

Commands:
    train                   Train model on given input data
    test                    Test model on given input data
    predict                 Predict for given data
    ofo                     Use online f-measure optimization
    version                 Print napkinXC version
    help                    Print help

Args:
    General:
    -i, --input             Input dataset, required
    -o, --output            Output (model) dir, required
    -m, --model             Model type (default = plt):
                            Models: ovr, br, hsm, plt, oplt, svbopFull, svbopHf,␣
↪brMips, svbopMips
    --ensemble              Number of models in ensemble (default = 1)
    -t, --threads           Number of threads to use (default = 0)
                            Note: -1 to use #cpus - 1, 0 to use #cpus
    --hash                  Size of features space (default = 0)
                            Note: 0 to disable hashing
    --featuresThreshold     Prune features below given threshold (default = 0.0)
    --seed                  Seed (default = system time)
    --verbose               Verbose level (default = 2)

    Base classifiers:
    --optimizer             Optimizer used for training binary classifiers (default =␣
↪liblinear)
                            Optimizers: liblinear, sgd, adagrad, fobos
    --bias                  Value of the bias features (default = 1)
    --weightsThreshold      Threshold value for pruning models weights (default = 0.1)

    LIBLINEAR:              (more about LIBLINEAR: https://github.com/cjlin1/
↪liblinear)
    -s, --liblinearSolver   LIBLINEAR solver (default for log loss = L2R_LR_DUAL, for␣
↪l2 loss = L2R_L2LOSS_SVC_DUAL)
                            Supported solvers: L2R_LR_DUAL, L2R_LR, L1R_LR,
                                               L2R_L2LOSS_SVC_DUAL, L2R_L2LOSS_SVC,␣
↪L2R_L1LOSS_SVC_DUAL, L1R_L2LOSS_SVC
    -c, --liblinearC        LIBLINEAR cost co-efficient, inverse of regularization␣
↪strength, must be a positive float,
                            smaller values specify stronger regularization (default =␣
↪10.0)
    --eps, --liblinearEps   LIBLINEAR tolerance of termination criterion (default = 0.
↪1)
```

(continues on next page)

(continued from previous page)

```
    SGD/AdaGrad:
    -l, --lr, --eta          Step size (learning rate) for online optimizers (default␣
↪= 1.0)
    --epochs                 Number of training epochs for online optimizers (default␣
↪= 1)
    --adagradEps             Defines starting step size for AdaGrad (default = 0.001)

    Tree:
    -a, --arity              Arity of tree nodes (default = 2)
    --maxLeaves              Maximum degree of pre-leaf nodes. (default = 100)
    --tree                   File with tree structure
    --treeType               Type of a tree to build if file with structure is not␣
↪provided
                             tree types: hierarchicalKmeans, huffman,␣
↪completeKaryInOrder, completeKaryRandom,
                                         balancedInOrder, balancedRandom,␣
↪onlineComplete

    K-Means tree:
    --kmeansEps              Tolerance of termination criterion of the k-means␣
↪clustering
                             used in hierarchical k-means tree building procedure␣
↪(default = 0.001)
    --kmeansBalanced         Use balanced K-Means clustering (default = 1)

    Prediction:
    --topK                   Predict top-k labels (default = 5)
    --threshold              Predict labels with probability above the threshold␣
↪(default = 0)
    --thresholds             Path to a file with threshold for each label

    Test:
    --measures               Evaluate test using set of measures (default = "p@1,r@1,
↪c@1,p@3,r@3,c@3,p@5,r@5,c@5")
                             Measures: acc (accuracy), p (precision), r (recall), c␣
↪(coverage), hl (hamming loos)
                                       p@k (precision at k), r@k (recall at k), c@k␣
↪(coverage at k), s (prediction size)
```

CHAPTER 3

Python API

3.1 Models

`models.PLT`
`models.HSM`
`models.BR`
`models.OVR`

3.2 Datasets

`datasets.download_dataset`
`datasets.load_dataset`
`datasets.load_libsvm_file`
`datasets.load_json_lines_file`
`datasets.to_csr_matrix`
`datasets.to_np_matrix`

3.3 Measures

`measures.precision_at_k`(Y_true, Y_pred[, k])	Calculate precision at 1-k places.
`measures.recall_at_k`(Y_true, Y_pred[, k, . . .])	Calculate recall at 1-k places.
`measures.coverage_at_k`(Y_true, Y_pred[, k])	Calculate coverage at 1-k places.
`measures.dcg_at_k`(Y_true, Y_pred[, k])	Calculate Discounted Cumulative Gain (DCG) at 1-k places.

Continued on next page

Table 3 – continued from previous page

measures.Jain_et_al_inverse_propensity(Y[, A, B])	Calculate inverse propensity as proposed in Jain et al.
measures.Jain_et_al_propensity(Y[, A, B])	Calculate propensity as proposed in Jain et al.
measures.ndcg_at_k(Y_true, Y_pred[, k, . . .])	Calculate normalized Discounted Cumulative Gain (nDCG) at 1-k places.
measures.psprecision_at_k(Y_true, Y_pred, inv_ps)	Calculate Propensity Scored Precision (PSP) at 1-k places.
measures.psrecall_at_k(Y_true, Y_pred, inv_ps)	Calculate Propensity Scored Recall (PSR) at 1-k places.
measures.psdcg_at_k(Y_true, Y_pred, inv_ps)	Calculate Propensity Scored Discounted Cumulative Gain (PSDCG) at 1-k places.
measures.psndcg_at_k(Y_true, Y_pred, inv_ps)	Calculate Propensity Scored normalized Discounted Cumulative Gain (PSnDCG) at 1-k places.
measures.hamming_loss(Y_true, Y_pred)	Calculate unnormalized (to avoid very small numbers because of large number of labels) hamming loss - average number of misclassified labels.
measures.f1_measure(Y_true, Y_pred[, . . .])	Calculate F1 measure, also known as balanced F-score or F-measure.

3.3.1 napkinxc.measures.precision_at_k

napkinxc.measures.**precision_at_k**(*Y_true*, *Y_pred*, *k=5*)

Calculate precision at 1-k places. Precision at k is defined as:

$$p@k = \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{\boldsymbol{y}})} y_l \,,$$

where $\boldsymbol{y} \in {0, 1}^m$ is ground truth label vector, $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ is predicted labels score vector, and $\text{rank}_k(\hat{\boldsymbol{y}})$ returns the k indices of $\hat{\boldsymbol{y}}$ with the largest values, ordered in descending order.

Parameters

- **Y_true** (*ndarray, csr_matrix, list[list|set[int|str]]*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **Y_pred** (*ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **k** (*int, optional*) – Calculate at places from 1 to k, defaults to 5

Returns Values of precision at 1-k places.

Return type ndarray

3.3.2 napkinxc.measures.recall_at_k

napkinxc.measures.**recall_at_k**(*Y_true*, *Y_pred*, *k=5*, *zero_division=0*)

Calculate recall at 1-k places. Recall at k is defined as:

$$r@k = \frac{1}{||\boldsymbol{y}||_1} \sum_{l \in \text{rank}_k(\hat{\boldsymbol{y}})} y_l \,,$$

where $\boldsymbol{y} \in 0, 1^m$ is ground truth label vector, $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ is predicted labels score vector, and $\text{rank}_k(\hat{\boldsymbol{y}})$ returns the k indices of $\hat{\boldsymbol{y}}$ with the largest values, ordered in descending order.

Parameters

- **`Y_true`** (*`ndarray, csr_matrix, list[list|set[int|str]]`*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*`ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]`*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **`k`** (*`int, optional`*) – Calculate at places from 1 to k, defaults to 5
- **`zero_division`** (*`float, optional`*) – Value to add when there is a zero division, typically set to 0, defaults to 0

Returns Values of recall at 1-k places.

Return type ndarray

3.3.3 napkinxc.measures.coverage_at_k

`napkinxc.measures.`**`coverage_at_k`**`(`*Y_true*, *Y_pred*, *k=5*`)`

Calculate coverage at 1-k places.

Parameters

- **`Y_true`** (*`ndarray, csr_matrix, list[list|set[int|str]]`*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*`ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]`*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **`k`** (*`int, optional`*) – Calculate at places from 1 to k, defaults to 5

Returns Values of coverage at 1-k places.

Return type ndarray

3.3.4 napkinxc.measures.dcg_at_k

`napkinxc.measures.`**`dcg_at_k`**`(`*Y_true*, *Y_pred*, *k=5*`)`

Calculate Discounted Cumulative Gain (DCG) at 1-k places. DCG at k is defined as:

$$DCG@k = \sum_{i=1}^{k} \frac{y_{\text{rank}_k(\hat{\boldsymbol{y}})_i}}{\log_2(i+1)} ,$$

where $\boldsymbol{y} \in 0, 1^m$ is ground truth label vector, $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ is predicted labels score vector, and $\text{rank}_k(\hat{\boldsymbol{y}})$ returns the k indices of $\hat{\boldsymbol{y}}$ with the largest values, ordered in descending order.

Parameters

- **Y_true** (*ndarray, csr_matrix, list[list|set[int|str]]*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **Y_pred** (*ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]]*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **k** (*int, optional*) – Calculate at places from 1 to k, defaults to 5

Returns Values of DCG at 1-k places.

Return type ndarray

3.3.5 napkinxc.measures.Jain_et_al_inverse_propensity

`napkinxc.measures.`**`Jain_et_al_inverse_propensity`**(*Y*, *A=0.55*, *B=1.5*)

Calculate inverse propensity as proposed in Jain et al. 2016. Inverse propensity q_l of label l is calculated as:

$$C = (\log N - 1)(B + 1)^A \, , \; q_l = 1 + C(N_l + B)^{-A} \, ,$$

where N is total number of data points, N_j is total number of data points for and A and B are dataset specific parameters.

Parameters

- **Y** (*ndarray, csr_matrix, list[list[tuple[int|str, float]]*) – Labels (typically ground truth for train data) provided as a matrix with non-zero values for relevant labels.
- **A** (*float, optional*) – Dataset specific parameter, typical values:
  - 0.5: `WikiLSHTC-325K` and `WikipediaLarge-500K`
  - 0.6: `Amazon-670K` and `Amazon-3M`
  - 0.55: otherwise

  Defaults to 0.55
- **B** (*float, optional*) – Dataset specific parameter, typical values:
  - 0.4: `WikiLSHTC-325K` and `WikipediaLarge-500K`
  - 2.6: `Amazon-670K` and `Amazon-3M`
  - 1.5: otherwise

  Defaults to 1.5

Returns Array with the inverse propensity for all labels

Return type ndarray

3.3.6 napkinxc.measures.Jain_et_al_propensity

`napkinxc.measures.`**`Jain_et_al_propensity`**(*Y*, *A=0.55*, *B=1.5*)

Calculate propensity as proposed in Jain et al. 2016. Propensity p_l of label l is calculated as:

$$C = (\log N - 1)(B + 1)^A \, , \; p_l = \frac{1}{1 + C(N_l + B)^{-A}} \, ,$$

where N is total number of data points, N_j is total number of data points for and A and B are dataset specific parameters.

Parameters

- **Y** (*ndarray, csr_matrix, list[list[int]]*) – Labels (typically ground truth for train data) provided as a matrix with non-zero values for relevant labels.
- **A** (*float, optional*) – Dataset specific parameter, typical values:
  - 0.5: `WikiLSHTC-325K` and `WikipediaLarge-500K`
  - 0.6: `Amazon-670K` and `Amazon-3M`
  - 0.55: otherwise

  Defaults to 0.55
- **B** (*float, optional*) – Dataset specific parameter, typical values:
  - 0.4: `WikiLSHTC-325K` and `WikipediaLarge-500K`
  - 2.6: `Amazon-670K` and `Amazon-3M`
  - 1.5: otherwise

  Defaults to 1.5

Returns Array with the propensity for all labels

Return type ndarray

3.3.7 napkinxc.measures.ndcg_at_k

`napkinxc.measures.`**`ndcg_at_k`**(*Y_true, Y_pred, k=5, zero_division=0*)

Calculate normalized Discounted Cumulative Gain (nDCG) at 1-k places.

Parameters

- **Y_true** (*ndarray, csr_matrix, list[list|set[int|str]]*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **Y_pred** (*ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **k** (*int, optional*) – Calculate at places from 1 to k, defaults to 5
- **zero_division** (*float, optional*) – Value to add when there is a zero division, typically set to 0, defaults to 0

Returns Values of nDCG at 1-k places.

Return type ndarray

3.3.8 napkinxc.measures.psprecision_at_k

`napkinxc.measures.`**`psprecision_at_k`**(*Y_true, Y_pred, inv_ps, k=5, normalize=True*)

Calculate Propensity Scored Precision (PSP) at 1-k places. This measure can be also called weighted precision.

PSP at k is defined as:

$$psp@k = \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{\boldsymbol{y}})} q_l \hat{y}_l,$$

where $\boldsymbol{y} \in 0, 1^m$ is ground truth label vector, $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ is predicted labels score vector, $\text{rank}_k(\hat{\boldsymbol{y}})$ returns the k indices of $\hat{\boldsymbol{y}}$ with the largest values, ordered in descending order, and $\boldsymbol{q}$ is vector of inverse propensities.

Parameters

- **`Y_true`** (*ndarray, csr_matrix, list[list|set[int|str]]*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]]*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **`inv_ps`** (*ndarray, list, dict*) – Inverse propensity (propensity scores) for each label (label weights). In case of text labels needs to be a dict.
- **`k`** (*int, optional*) – Calculate at places from 1 to k, defaults to 5
- **`normalize`** (*bool, optional*) – Normalize result to [0, 1] range by dividing it by best possible value, commonly used to report results, defaults to True

Returns Values of PSP at 1-k places.

Return type ndarray

3.3.9 napkinxc.measures.psrecall_at_k

`napkinxc.measures.`**`psrecall_at_k`**(*Y_true, Y_pred, inv_ps, k=5, normalize=True, zero_division=0*)

Calculate Propensity Scored Recall (PSR) at 1-k places.

Parameters

- **`Y_true`** (*ndarray, csr_matrix, list[list|set[int|str]]*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]]*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **`inv_ps`** (*ndarray, list, dict*) – Inverse propensity (propensity scores) for each label. In case of text labels needs to be a dict.
- **`k`** (*int, optional*) – Calculate at places from 1 to k, defaults to 5
- **`zero_division`** (*float, optional*) – Value to add when there is a zero division, typically set to 0, defaults to 0
- **`normalize`** (*bool, optional*) – Normalize result to [0, 1] range by dividing it by best possible value, commonly used to report results, defaults to True

Returns Values of PSR at 1-k places.

Return type ndarray

3.3.10 napkinxc.measures.psdcg_at_k

`napkinxc.measures.`**`psdcg_at_k`**(*Y_true*, *Y_pred*, *inv_ps*, *k=5*, *normalize=True*)

Calculate Propensity Scored Discounted Cumulative Gain (PSDCG) at 1-k places.

Parameters

- **`Y_true`** (*`ndarray, csr_matrix, list[list|set[int|str]]`*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*`ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]`*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **`inv_ps`** (*`ndarray, list, dict`*) – Inverse propensity (propensity scores) for each label. In case of text labels needs to be a dict.
- **`k`** (*`int, optional`*) – Calculate at places from 1 to k, defaults to 5
- **`normalize`** (*`bool, optional`*) – Normalize result to [0, 1] range by dividing it by best possible value, commonly used to report results, defaults to True

Returns Values of PSDCG at 1-k places.

Return type ndarray

3.3.11 napkinxc.measures.psndcg_at_k

`napkinxc.measures.`**`psndcg_at_k`**(*Y_true*, *Y_pred*, *inv_ps*, *k=5*, *zero_division=0*, *normalize=True*)

Calculate Propensity Scored normalized Discounted Cumulative Gain (PSnDCG) at 1-k places.

Parameters

- **`Y_true`** (*`ndarray, csr_matrix, list[list|set[int|str]]`*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*`ndarray, csr_matrix, list[list[int|str]], list[list[tuple[int|str, float]]`*) – Predicted labels provided as a matrix with scores or list of rankings as a list of labels or tuples of labels with scores (label, score). In the case of the matrix, the ranking will be calculated by sorting scores in descending order.
- **`inv_ps`** (*`ndarray, list, dict`*) – Inverse propensity (propensity scores) for each label. In case of text labels needs to be a dict.
- **`k`** (*`int, optional`*) – Calculate at places from 1 to k, defaults to 5
- **`zero_division`** (*`float, optional`*) – Value to add when there is a zero division, typically set to 0, defaults to 0
- **`normalize`** (*`bool, optional`*) – Normalize result to [0, 1] range by dividing it by best possible value, commonly used to report results, defaults to True

Returns Values of PSnDCG at 1-k places.

Return type ndarray

3.3.12 napkinxc.measures.hamming_loss

`napkinxc.measures.`**`hamming_loss`**(*Y_true*, *Y_pred*)

Calculate unnormalized (to avoid very small numbers because of large number of labels) hamming loss - average number of misclassified labels.

Parameters

- **`Y_true`** (*`ndarray, csr_matrix, list[list|set[int|str]]`*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*`ndarray, csr_matrix, list[list|set[int|str]], list[list|set[tuple[int|str, float]]`*) – Predicted labels provided as a matrix with scores or list of lists of labels or tuples of labels with scores (label, score).

Returns Value of hamming loss.

Return type float

3.3.13 napkinxc.measures.f1_measure

`napkinxc.measures.`**`f1_measure`**(*Y_true*, *Y_pred*, *average='micro'*, *zero_division=0*)

Calculate F1 measure, also known as balanced F-score or F-measure.

Parameters

- **`Y_true`** (*`ndarray, csr_matrix, list[list|set[int|str]]`*) – Ground truth provided as a matrix with non-zero values for true labels or a list of lists or sets of true labels.
- **`Y_pred`** (*`ndarray, csr_matrix, list[list|set[int|str]], list[list|set[tuple[int|str, float]]`*) – Predicted labels provided as a matrix with scores or list of lists of labels or tuples of labels with scores (label, score).
- **`average`** (*`str`*) – Determines the type of performed averaging {`'micro'`, `'macro'`, `'sample'`}, default to `'micro'`
- **`zero_division`** (*`float, optional`*) – Value to add when there is a zero division, typically set to 0, defaults to 0

Returns Value of F1-measure.

Return type float

CHAPTER 4

Indices and tables

- genindex
- search

Index