Requirements

conda install numpy scipy scikit-learn pandas joblib pytorch

pip install tpot

expectation

* run long time
* diff solution every time runs it

usage

**from** tpot **import** TPOTClassifier

pipeline\_optimizer = TPOTClassifier()

pipeline\_optimizer = TPOTClassifier(generations=5, population\_size=20, cv=5,

random\_state=42, verbosity=2) pipeline\_optimizer.fit(X\_train, y\_train)

print(pipeline\_optimizer.score(X\_test, y\_test))

pipeline\_optimizer.export('tpot\_exported\_pipeline.py')

best code pipeline

**from** tpot **import** TPOTClassifier

**from** sklearn.datasets **import** load\_digits

**from** sklearn.model\_selection **import** train\_test\_split

digits = load\_digits()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, digits.target,

train\_size=0.75, test\_size=0.25)

pipeline\_optimizer = TPOTClassifier(generations=5, population\_size=20, cv=5,

random\_state=42, verbosity=2)

pipeline\_optimizer.fit(X\_train, y\_train)

print(pipeline\_optimizer.score(X\_test, y\_test))

pipeline\_optimizer.export('tpot\_exported\_pipeline.py')

Detailed descriptions of the command-line arguments are below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Argument** | **Parameter** | **Valid values** | **Effect** |
| -is | INPUT\_SEPARATOR | Any string | Character used to separate columns in the input file. |
| -target | TARGET\_NAME | Any string | Name of the target column in the input file. |
| -mode | TPOT\_MODE | ['classification', 'regression'] | Whether TPOT is being used for a supervised classification or regression problem. |
| -o | OUTPUT\_FILE | String path to a file | File to export the code for the final optimized pipeline. |
| -g | GENERATIONS | Any positive integer or None | Number of iterations to run the pipeline optimization process. It must be a positive number or None. If None, the parameter max\_time\_mins must be defined as the runtime limit. Generally, TPOT will work better when you give it more generations (and therefore time) to optimize the pipeline.  TPOT will evaluate POPULATION\_SIZE + GENERATIONS x OFFSPRING\_SIZE pipelines in total. |
| -p | POPULATION\_SIZE | Any positive integer | Number of individuals to retain in the GP population every generation. Generally, TPOT will work better when you give it more individuals (and therefore time) to optimize the pipeline.  TPOT will evaluate POPULATION\_SIZE + GENERATIONS x OFFSPRING\_SIZE pipelines in total. |
| -os | OFFSPRING\_SIZE | Any positive integer | Number of offspring to produce in each GP generation.  By default, OFFSPRING\_SIZE = POPULATION\_SIZE. |
| -mr | MUTATION\_RATE | [0.0, 1.0] | GP mutation rate in the range [0.0, 1.0]. This tells the GP algorithm how many pipelines to apply random changes to every generation.  We recommend using the default parameter unless you understand how the mutation rate affects GP algorithms. |
| -xr | CROSSOVER\_RATE | [0.0, 1.0] | GP crossover rate in the range [0.0, 1.0]. This tells the GP algorithm how many pipelines to "breed" every generation.  We recommend using the default parameter unless you understand how the crossover rate affects GP algorithms. |
| -scoring | SCORING\_FN | 'accuracy', 'adjusted\_rand\_score', 'average\_precision', 'balanced\_accuracy', 'f1', 'f1\_macro', 'f1\_micro', 'f1\_samples', 'f1\_weighted', 'neg\_log\_loss', 'neg\_mean\_absolute\_error', 'neg\_mean\_squared\_error', 'neg\_median\_absolute\_error', 'precision', 'precision\_macro', 'precision\_micro', 'precision\_samples', 'precision\_weighted', 'r2', 'recall', 'recall\_macro', 'recall\_micro', 'recall\_samples', 'recall\_weighted', 'roc\_auc', 'my\_module.scorer\_name\*' | Function used to evaluate the quality of a given pipeline for the problem. By default, accuracy is used for classification and mean squared error (MSE) is used for regression.  TPOT assumes that any function with "error" or "loss" in the name is meant to be minimized, whereas any other functions will be maximized.  my\_module.scorer\_name: You can also specify your own function or a full python path to an existing one.  See the section on [scoring functions](http://epistasislab.github.io/tpot/using/#scoring-functions) for more details. |
| -cv | CV | Any integer > 1 | Number of folds to evaluate each pipeline over in k-fold cross-validation during the TPOT optimization process. |
| -sub | SUBSAMPLE | (0.0, 1.0] | Subsample ratio of the training instance. Setting it to 0.5 means that TPOT randomly collects half of training samples for pipeline optimization process. |
| -njobs | NUM\_JOBS | Any positive integer or -1 | Number of CPUs for evaluating pipelines in parallel during the TPOT optimization process.  Assigning this to -1 will use as many cores as available on the computer. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Thus for n\_jobs = -2, all CPUs but one are used. |
| -maxtime | MAX\_TIME\_MINS | Any positive integer | How many minutes TPOT has to optimize the pipeline.  How many minutes TPOT has to optimize the pipeline.If not None, this setting will allow TPOT to run until max\_time\_mins minutes elapsed and then stop. TPOT will stop earlier if generationsis set and all generations are already evaluated. |
| -maxeval | MAX\_EVAL\_MINS | Any positive float | How many minutes TPOT has to evaluate a single pipeline.  Setting this parameter to higher values will allow TPOT to consider more complex pipelines but will also allow TPOT to run longer. |
| -s | RANDOM\_STATE | Any positive integer | Random number generator seed for reproducibility.  Set this seed if you want your TPOT run to be reproducible with the same seed and data set in the future. |
| -config | CONFIG\_FILE | String or file path | Operators and parameter configurations in TPOT:   * Path for configuration file: TPOT will use the path to a configuration file for customizing the operators and parameters that TPOT uses in the optimization process * string 'TPOT light', TPOT will use a built-in configuration with only fast models and preprocessors * string 'TPOT MDR', TPOT will use a built-in configuration specialized for genomic studies * string 'TPOT sparse': TPOT will use a configuration dictionary with a one-hot encoder and the operators normally included in TPOT that also support sparse matrices.   See the [built-in configurations](http://epistasislab.github.io/tpot/using/#built-in-tpot-configurations) section for the list of configurations included with TPOT, and the [custom configuration](http://epistasislab.github.io/tpot/using/#customizing-tpots-operators-and-parameters) section for more information and examples of how to create your own TPOT configurations. |
| -template | TEMPLATE | String | Template of predefined pipeline structure. The option is for specifying a desired structure for the machine learning pipeline evaluated in TPOT. So far this option only supports linear pipeline structure. Each step in the pipeline should be a main class of operators (Selector, Transformer, Classifier or Regressor) or a specific operator (e.g. `SelectPercentile`) defined in TPOT operator configuration. If one step is a main class, TPOT will randomly assign all subclass operators (subclasses of [`SelectorMixin`](https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature\_selection/base.py#L17), [`TransformerMixin`](https://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html), [`ClassifierMixin`](https://scikit-learn.org/stable/modules/generated/sklearn.base.ClassifierMixin.html) or [`RegressorMixin`](https://scikit-learn.org/stable/modules/generated/sklearn.base.RegressorMixin.html) in scikit-learn) to that step. Steps in the template are delimited by "-", e.g. "SelectPercentile-Transformer-Classifier". By default value of template is None, TPOT generates tree-based pipeline randomly. See the [template option in tpot](http://epistasislab.github.io/tpot/using/#template-option-in-tpot) section for more details. |
| -memory | MEMORY | String or file path | If supplied, pipeline will cache each transformer after calling fit. This feature is used to avoid computing the fit transformers within a pipeline if the parameters and input data are identical with another fitted pipeline during optimization process. Memory caching mode in TPOT:   * Path for a caching directory: TPOT uses memory caching with the provided directory and TPOT does NOT clean the caching directory up upon shutdown. * string 'auto': TPOT uses memory caching with a temporary directory and cleans it up upon shutdown. |
| -cf | CHECKPOINT\_FOLDER | Folder path | If supplied, a folder you created, in which tpot will periodically save pipelines in pareto front so far while optimizing.  This is useful in multiple cases:   * sudden death before tpot could save an optimized pipeline * progress tracking * grabbing a pipeline while tpot is working   Example: mkdir my\_checkpoints -cf ./my\_checkpoints |
| -es | EARLY\_STOP | Any positive integer | How many generations TPOT checks whether there is no improvement in optimization process.  End optimization process if there is no improvement in the set number of generations. |
| -v | VERBOSITY | {0, 1, 2, 3} | How much information TPOT communicates while it is running.  0 = none, 1 = minimal, 2 = high, 3 = all.  A setting of 2 or higher will add a progress bar during the optimization procedure. |
| -log | LOG | Folder path | Save progress content to a file. |
| --no-update-check | | | Flag indicating whether the TPOT version checker should be disabled. |
| --version | | | Show TPOT's version number and exit. |
| --help | | | Show TPOT's help documentation and exit. |

custom scoring

**ef** **my\_custom\_accuracy**(y\_true, y\_pred):

**return** float(sum(y\_pred == y\_true)) / len(y\_true)

*# Make a custom a scorer from the custom metric function*

*# Note: greater\_is\_better=False in make\_scorer below would mean that the scoring function should be minimized.*

my\_custom\_scorer = make\_scorer(my\_custom\_accuracy, greater\_is\_better=**True**)

tpot = TPOTClassifier(generations=5, population\_size=20, verbosity=2,

scoring=my\_custom\_scorer)

built-in

TPOT comes with a handful of default operators and parameter configurations that we believe work well for optimizing machine learning pipelines. Below is a list of the current built-in configurations that come with TPOT.

|  |  |  |
| --- | --- | --- |
| **Configuration Name** | **Description** | **Operators** |
| Default TPOT | TPOT will search over a broad range of preprocessors, feature constructors, feature selectors, models, and parameters to find a series of operators that minimize the error of the model predictions. Some of these operators are complex and may take a long time to run, especially on larger datasets.  **Note: This is the default configuration for TPOT.** To use this configuration, use the default value (None) for the config\_dict parameter. | [Classification](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier.py)  [Regression](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/regressor.py) |
| TPOT light | TPOT will search over a restricted range of preprocessors, feature constructors, feature selectors, models, and parameters to find a series of operators that minimize the error of the model predictions. Only simpler and fast-running operators will be used in these pipelines, so TPOT light is useful for finding quick and simple pipelines for a classification or regression problem.  This configuration works for both the TPOTClassifier and TPOTRegressor. | [Classification](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier_light.py)  [Regression](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/regressor_light.py) |
| TPOT MDR | TPOT will search over a series of feature selectors and [Multifactor Dimensionality Reduction](https://en.wikipedia.org/wiki/Multifactor_dimensionality_reduction) models to find a series of operators that maximize prediction accuracy. The TPOT MDR configuration is specialized for [genome-wide association studies (GWAS)](https://en.wikipedia.org/wiki/Genome-wide_association_study), and is described in detail online [here](https://arxiv.org/abs/1702.01780).  Note that TPOT MDR may be slow to run because the feature selection routines are computationally expensive, especially on large datasets. | [Classification](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier_mdr.py)  [Regression](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/regressor_mdr.py) |
| TPOT sparse | TPOT uses a configuration dictionary with a one-hot encoder and the operators normally included in TPOT that also support sparse matrices.  This configuration works for both the TPOTClassifier and TPOTRegressor. | [Classification](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier_sparse.py)  [Regression](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/regressor_sparse.py) |
| TPOT NN | TPOT uses the same configuration as "Default TPOT" plus additional neural network estimators written in PyTorch (currently only `tpot.builtins.PytorchLRClassifier` and `tpot.builtins.PytorchMLPClassifier`).  Currently only classification is supported, but future releases will include regression estimators. | [Classification](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier_nn.py) |
| TPOT cuML | TPOT will search over a restricted configuration using the GPU-accelerated estimators in [RAPIDS cuML](https://github.com/rapidsai/cuml) and [DMLC XGBoost](https://github.com/dmlc/xgboost). This configuration requires an NVIDIA Pascal architecture or better GPU with compute capability 6.0+, and that the library cuML is installed. With this configuration, all model training and predicting will be GPU-accelerated.  This configuration is particularly useful for medium-sized and larger datasets on which CPU-based estimators are a common bottleneck, and works for both the TPOTClassifier and TPOTRegressor. | [Classification](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier_cuml.py)  [Regres](https://github.com/EpistasisLab/tpot/blob/master/tpot/config/regressor_cuml.py) |

**from** tpot **import** TPOTClassifier

**from** sklearn.datasets **import** load\_digits

**from** sklearn.model\_selection **import** train\_test\_split

digits = load\_digits()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, digits.target,

train\_size=0.75, test\_size=0.25)

tpot = TPOTClassifier(generations=5, population\_size=20, verbosity=2,

config\_dict='TPOT light')

tpot.fit(X\_train, y\_train)

print(tpot.score(X\_test, y\_test))

tpot.export('tpot\_digits\_pipeline.py')

pipeline template

tpot\_obj = TPOTClassifier(

template='Selector-Transformer-Classifier'

)

select features – fit and transform – scoring classify

pipeline caching – to avoid similar config and repeated computations

**from** tpot **import** TPOTClassifier

**from** tempfile **import** mkdtemp

**from** joblib **import** Memory

**from** shutil **import** rmtree

*# Method 1, auto mode: TPOT uses memory caching with a temporary directory and cleans it up upon shutdown*

tpot = TPOTClassifier(memory='auto')

*# Method 2, with a custom directory for memory caching*

tpot = TPOTClassifier(memory='/to/your/path')

*# Method 3, with a Memory object*

cachedir = mkdtemp() *# Create a temporary folder*

memory = Memory(cachedir=cachedir, verbose=0)

tpot = TPOTClassifier(memory=memory)

*# Clear the cache directory when you don't need it anymore*

rmtree(cachedir)

**Telling TPOT to use built-in PyTorch neural network models**

Mainly due to the issues described below, TPOT won't use its neural network models unless you explicitly tell it to do so. This is done as follows:

* Use import tpot.nn before instantiating any TPOT estimators.
* Use a configuration dictionary that includes one or more tpot.nn estimators, either by writing one manually, including one from a file, or by importing the configuration in tpot/config/classifier\_nn.py. A very simple example that will force TPOT to only use a PyTorch-based logistic regression classifier as its main estimator is as follows:

tpot\_config = {

'tpot.nn.PytorchLRClassifier': {

'learning\_rate': [1e-3, 1e-2, 1e-1, 0.5, 1.]

}

}

boston house prediction

<http://lib.stat.cmu.edu/datasets/boston>

https://www.kaggle.com/puxama/bostoncsv

<https://www.kaggle.com/prasadperera/the-boston-housing-dataset>

bank marketing data set

https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

https://notebook.community/teaearlgraycold/tpot/tutorials/Portuguese%20Bank%20Marketing/Portuguese%20Bank%20Marketing%20Stratergy

<https://heartbeat.fritz.ai/data-handling-scenarios-part-2-working-with-missing-values-in-a-dataset-34b758cfc9fa>

<https://github.com/EpistasisLab/tpot/issues/511>

<https://arxiv.org/pdf/1706.01120.pdf>

Diagram

Description automatically generated

**Attribute Information:**

Input variables:  
# bank client data:  
1 - age (numeric)  
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')  
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)  
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')  
5 - default: has credit in default? (categorical: 'no','yes','unknown')  
6 - housing: has housing loan? (categorical: 'no','yes','unknown')  
7 - loan: has personal loan? (categorical: 'no','yes','unknown')  
# related with the last contact of the current campaign:  
8 - contact: contact communication type (categorical: 'cellular','telephone')  
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')  
10 - day\_of\_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')  
11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.  
# other attributes:  
12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)  
13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)  
14 - previous: number of contacts performed before this campaign and for this client (numeric)  
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')  
# social and economic context attributes  
16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)  
17 - cons.price.idx: consumer price index - monthly indicator (numeric)  
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)  
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)  
20 - nr.employed: number of employees - quarterly indicator (numeric)  
  
Output variable (desired target):  
21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

1. CRIM - per capita crime rate by town
2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS - proportion of non-retail business acres per town.
4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX - nitric oxides concentration (parts per 10 million)
6. RM - average number of rooms per dwelling
7. AGE - proportion of owner-occupied units built prior to 1940
8. DIS - weighted distances to five Boston employment centres
9. RAD - index of accessibility to radial highways
10. TAX - full-value property-tax rate per $10,000
11. PTRATIO - pupil-teacher ratio by town
12. B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT - % lower status of the population
14. MEDV - Median value of owner-occupied homes in $1000's