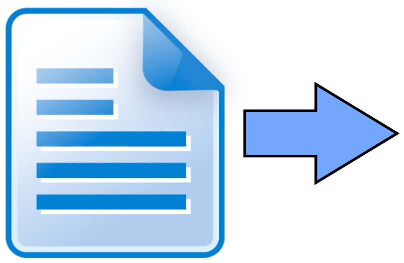
GFTCT description

This document is intended to describe *General full-text classification tool-set* (GFTCT). Its principles of operation, usage workflow and underlaying methods. Last part of the document describes program structure more detailed and from implementation point of view.

General

GFTCT is designed to make classification of public company announcements, but in principle can be used for other classification tasks. Announcements are represented as sets of words from announcement text without taking into account their order in text.



word	count
conversation	2
pointless	1
mainly	1
basic	3
...	...

So the hypothesis is that we can make a trading decision about corresponding company stocks (wether to buy stocks or stock-options or sell them) based on such representation of announcements and these decisions will give us positive mean profit.

To check this hypothesis we use set of announcements with known profitability thus they could be divided into two classes «pos» and «neg» and then part of these announcements could be used for training and other part for testing and thus test whether this approach effective or not. But from implementation point of view there are some pitfalls:

- in order to use classification techniques samples should be represented as fixed length vectors, how can we vectorize word-count sets?
- how could we be sure that it was not just a luck with test set prediction results?
- if we will have say 60% correct prediction of class how it affect profit? (we could had 60% correct results with low profit and 40% of incorrect with big loss)

Next sections describe how these questions are resolved in GFTCT.

Vectorization

In order to vectorize announcements first we should fix vector length. Since we represent announcements as set of words the length of vectors is determined by dictionary length. But we can not use ordinary dictionary with all possible english words because in such case vectors length will be more than 200,000 and it will be very difficult for classification algorithms to work with such huge amount of data (200,000 words x 10,000 announcements is very big matrix).

The decision is to use only words from announcements with some additional restrictions. And from this moment we should understand that if we use some announcements for dictionary creation we can not use them as test samples. Because in testing we should model situation when we are trying to classify new announcement, but we had no this announcement when we were training our classifiers or creating our dictionary for vectorization. So announcements used for generation of vectorization scheme can not be used as test samples!
With GFTCT you determine set of announcements which will be used for vectorization scheme generation this scheme is called **transformation**. It is possible to create plenty of different transformations, prediction performance strongly depend on how it is done. Transformation should be created on rather representative set of announcements.

Transformation generation steps and configurable options (when «announcement/s» used in table it is considered only announcements selected for transformation generation):

step	what may be configured
1. Create a dictionary	- consider negations (whether phrase like «not good» should be represented as one dictionary item, or two: «not» and «good» - filter stop words (remove words like: and, do, etc. because they are very widespread and not representative) - filter short words (it is possible to determine minimal acceptable word length)
2. Reduce dictionary size	- remove words that occurred only in N and less documents (it is obvious that if word occurred only in one document it can not be used as vector element because learning algorithms should find some generalization and it is impossible to generalize based on only one case, two cases for large data amount is not enough too, but from other side if word appeared in every document it can not help in generalization too, so the golden mean should be found) - take top-X words (as soon as announcements are marked with «pos» and «neg» markers it is possible to calculate frequency of every word among «pos» announcements and «neg» announcements if these frequencies differ not very much for certain word then this word is not very representative for pos/neg discrimination. But if it's appeared that word «nice» occurred in 15 «pos»-marked documents and in 2 «neg»-marked then maybe this word can help us more in classification then say word «and» (this is just a hypothesis but rather feasible and it can be used for dictionary reduction). 17/2 is better then 13/3 but as good as 2/17 because 2/17 helps us in «neg» classification as much as 17/2 helps in «pos» classification. So words sorted in descending order of abs(log(p/n)) because log(p/n) = -log(n/p) so absolute values are equal and the more p differs from n (or vice versa) the higher rank word will have), it is possible not to use this option and take all words obtained after previous filters
3. Chose representation	- represent announcements as vectors where first component corresponds to occurrence of first dictionary word, second - to occurrence of second dictionary word, etc., so vector consists of 0 and 1, 0 - if there is no such word in document, 1 - there is one or more such words in document, so vectors are boolean vectors - same as previous, but instead of ones put the word frequency in document - same as previous, but also multiply frequency by idf (inverse document frequency) so if word is in many documents its weight decreases if word is rather rare among documents its weight increases - this is so called tf-idf transform

step	what may be configured
4. Make PCA	After Step 3 we already have vectors because dictionary fixed and representation is chosen. But length of these vectors can be still high, so it is possible to make further reduction of dimension using principal component analysis (PCA, it is implemented with randomized SVD technique). On this step only structure of vectors is taken into account «pos» and «neg» labels are ignored. PCA is intended to reduce dimension with minimal loss of information, so if you make PCA transformation on vectors with length say 10,000 make their length 500 then if you make reverse transformation on any of these 500-d vectors back to 10,000-d you will have mostly the same 10,000 vector with minimal difference. But 10,000-d vector has dictionary correspondence: first component correspond to first word in dictionary and so forth, 500-d vector can not be intuitively interpreted. You may chose to use or not this reduction and how many new components should be left.
5. Save transformation	As transformation generation can be not very fast it is useful to save it once generated and use for further sample vectorization. Transformation has different options and may be based on different announcement sets, so there is a place for experiments to find optimal announcement representation for better classification.

Cross-validation
When announcements are vectorized by some transformation they are marked if they were used for transformation generation and thus they will not be used as test samples, so for testing some extra announcements should be provided. These extra announcements should have approximately the same distribution of «pos» / «neg» samples and approximately the same distribution of symbols (company markers) as transformation samples.
To determine if our learner good or not we should try to train it with training samples (training samples could be selected from transformation samples and from other samples) and them try to predict classes (pos/neg) of test samples. To be sure that test performance is correct we should do it for several times.
Standard approach in machine-learning is K-fold stratified cross-validation. It means that we take our samples and divide the into K subsets with approximately the same distribution of labels (pos/neg), then we use first set as test and all rest samples as training, train predictors, test them, remember results and after that we take next set to use it as test and again all other samples are used as training and do so K-times.
In such way we obtain K results and average then. These averaged results are rather good estimation of performance of the predictor.
So the idea is that our approach: vectorization scheme + learning algorithm should provide certain results.
K is commonly selected to be 10 if we have enough samples.
In principle for cross-validation block just samples are used as input (announcements in our case), so transformation could be based on training set and it is ok, but transformation could be rather long process that’s why in GFTCT it is excluded from cross-validation block -- samples first transformed (vectorized) and then tested.
So in GFTCT all samples that was provided to training module and was not exposed to participate in transformation creation are divided into 10 test sets. Then 10 training/test

experiments are conducted, where as training set all samples except current test set are used.
Then averaged results are output.
In GFTCT five different classifiers are used. All these classifiers have only ‘pos’ or ‘neg’ as output. But also three additional ensemble classifiers are implemented they use prediction results of these five and make ensemble prediction. First makes prediction of majority of 5 classifiers, so it outputs only «pos» or «neg» too (because five is odd). Second makes prediction based on majority’s opinion if only one or less has opposite opinion, this classifier sometimes may have no decision for example in the case then 3 predict «pos» and 2 «neg». And the last (third) ensemble classifier makes prediction if and only if all 5 predict unanimously.
There are also 3 «predictors» which can be only used for comparison of results: «predictor» which always predict «pos», «predictor» which always predict «neg» and «predictor» which always choose random answer («pos» or «neg»). These predictors are only for comparison and they don’t participate in ensembles.
If results are appropriate then our approach is correct. So we can use all samples (with certain symbols if we had made such filtration) for transformation creation and then all of them for learning and resulting model supposed to make results comparable with experimental ones.

Using samples weights
Our idea is to improve trading. So if we can predict many announcement effects but in cases of errors we will have aggregated losses higher than profits when we made correct predictions this predictor is useless.
Every announcement should have not only class label (pos/neg) but also two estimations: loss in the case of error prediction and profit in the case of correct prediction. These estimations could be done in many ways and as first approximation they could be equal and both equal to absolute value of the difference between next day open and next day close. But they could be calculated in more sophisticated way with considering intraday price dynamic, position opening/closing, stops trailing strategy and risk management strategy.
Training/test module can take into account sample weights so when making training/test weights could be used. It is possible to use profit as weights or loss, in first case classifier tries to make predictions so that profit is maximized, in second case classifier learns to minimize possible losses. These approaches may lead to different results if you have asymmetric profit/loss estimations, this asymmetry appears as soon as you only take into account difference between bid and ask prices to say nothing of more sophisticated profit/loss estimation approaches.
So not only precision of pos/neg classification is estimated in training/test module, but also mean expected profit/loss.
It should be mentioned, that sample labels and profit/loss estimations could be changed after transformation is generated and samples are transformed (vectorized). So there is no need to repeat all transformation and sample generation if you only want to change these estimations.

Application installation

To run GFTCT you need to have installed GhostScript, Python 2.7 and python modules:

- SQLObject
- xlrd
- dpfminer
- PyYAML
- lxml
- nltk
- numpy
- scipy
- scikit-learn

If you run on Windows you should add to PATH definition path to python installation and path to python/Scripts, when I tested I've installed python to c:\Python27 so I had to add to PATH c:\Python27;c:\Python27\Scripts.
Most difficult to install from source are last five. It is better to install them as binary packages. For Windows system you may just use following links to binary installators, you should run them after python installation:

Numpy
http://sourceforge.net/projects/numpy/files/NumPy/1.6.1/numpy-1.6.1-win32-superpack-python2.7.exe/download

Scipy
http://sourceforge.net/projects/scipy/files/scipy/0.10.0/scipy-0.10.0-win32-superpack-python2.7.exe/download

NLTK
http://nltk.googlecode.com/files/nltk-2.0b9.win32.exe

Scikit-learn
http://sourceforge.net/projects/scikit-learn/files/scikit-learn-0.9.win32-py2.7.exe/download

lxml
http://pypi.python.org/packages/2.7/l/lxml/lxml-2.2.8.win32-py2.7.exe

You may also use these packages as binary too:
xlrd
http://pypi.python.org/packages/any/x/xlrd/xlrd-0.7.1.win32.exe

PyYAML
http://pypi.python.org/packages/2.7/P/PyYAML/PyYAML-3.10.win32-py2.7.exe

Other packages you may install with setuptools or pip. For Windows system I recommend setuptools:

http://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11.win32-py2.7.exe

After you installed setuptools, you have just type in command line:

```
easy_install sqlobject
```

```
easy_install pdfminer
```

Finally if you want to use pdf announcements you should install GhostScript:
http://www.ghostscript.com/download/gsdnld.html
It is better to install it to some directory with no spaces in the name (for example c:\gs, not to c:\Program Files\gs) so you'll have less problems with the path to this application later.

GFTCT has some PC-specific parameters you should tune in app_libs/app_config.py. It is very easy to tune parameters, app_config.py is just Python script with constants:

```
GS_PATH = "c:/gs/gs9.04/bin/gswin32"

XLS_DATE_MODE = 0 # 0 - windows, 1 - mac
CSV_DATE_PATTERN = "%m.%d.%Y"
CONFIG_DATE_PATTERN = "%m/%d/%Y"

DB_TYPE = "sqlite"
DB_PATH = "./data/data.sqlite"
FILES_PATH = "./data/files"

CSV_DELIMITER_WRITE = ";"
CSV_DELIMITER_READ = ";"

TMP_PDF = "temporary.pdf"
```

You may need to change one date format (CSV_DATE_PATTERN if you use not 4 but 2 digits for year in CSV use pattern "%m.%d.%y" or if you use slashes instead of dots change them to slashes) and GS_PATH. Maybe you'll need to change csv delimiters - they are system specific (sometimes commas, sometimes semicolons) on my Mac Excel easily opens csv with comma delimiter, but if semicolon is used Excel doesn't split columns and on Win machine I have opposite situation.

Now you have to go to application directory and run:

```
python app_setup.py
```

This command prepares DB file, downloads data for text-processing and creates test csv file with movie review database so you may test application.

Test run

When application is installed you may test it. File MOVIE.csv has necessary structure to be given to prepare.py:

```
python prepare.py import MOVIE.csv --no-copy
```

and then

```
python prepare.py tokenize
```

File smp.cfg is sample file with transformation configuration for movie review db, it uses only 200 samples for fast results. Run:

```
python transform.py smp.cfg --csv-report dict.csv
```

As the result three files will be created: dict.csv with information about words in dictionary, samples.pickle with samples and movie-short.trans.pickle with transformation. Last two are binary files and can not be inspected. But you may inspect smp.cfg itself and dict.csv for better understanding.
Now we have samples ready to be learned and tested:

```
python learn.py samples.pickle --save-model model.pickle
```

Testing results will be output and prediction model will be saved as model.pickle
Now you may take some sample from MOVIE.csv with symbol «MOVIE» (only samples with symbol «MOVIE_SHORT» are used in smp.cfg) - just copy path and paste it as parameter of predict.py:

```
python predict.py model.pickle <path> --report
```

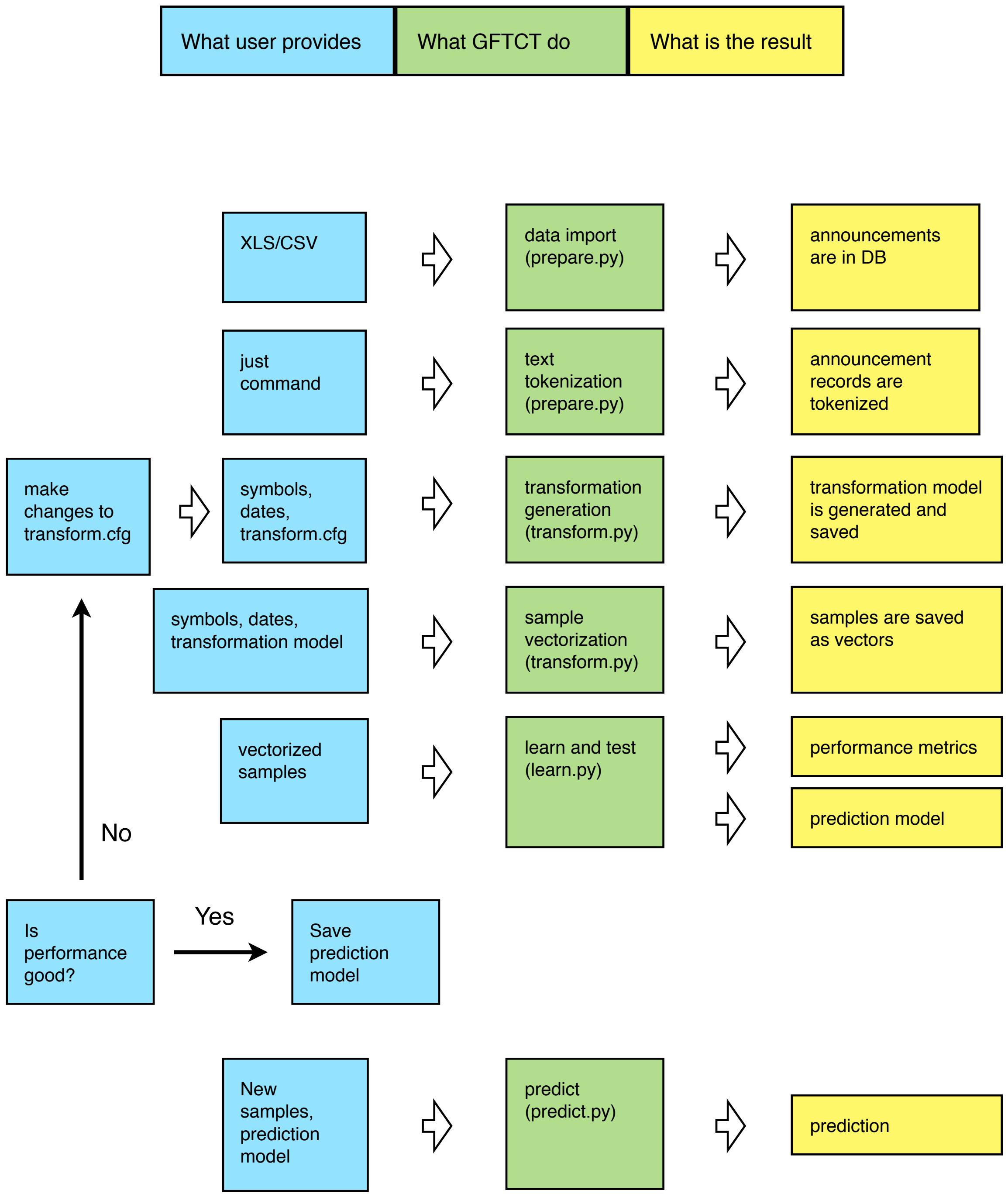
You'll obtain prediction (maybe not very good as we used just 200 samples for training) and report.csv with words from file and dictionary.
Try to make some changes to *smp.cfg -> transform -> learn* to see results.

GFTCT usage workflow and commands description

There are four main tools within GFTCT:

Command name	Purpose
prepare.py	- data import - data tokenization - change of announcement class labels and associated profit/loss
transform.py	- prepare transformation - transform (vectorize) samples
learn.py	- train on samples and make tests
predict.py	- make predictions for new announcements

GFTCT is intended to work with announcements so it affected workflow scheme.
In short standard workflow is the following:


What user provides
What GFTCT do
What is the result
XLS/CSV
data import (prepare.py)
announcements are in DB
just command
text tokenization (prepare.py)
announcement records are tokenized
make changes to transform.cfg
symbols, dates, transform.cfg
transformation generation (transform.py)
transformation model is generated and saved
symbols, dates, transformation model
sample vectorization (transform.py)
samples are saved as vectors
vectorized samples
learn and test (learn.py)
performance metrics
prediction model
No
Is performance good?
Yes
Save prediction model
New samples, prediction model
predict (predict.py)
prediction

prepare.py
This tool allows initial data import and subsequent manipulation. All announcements in GFTCT are recorded in sqlite database. As initially announcements may be in different formats (txt, html, pdf) there is standard input format for big lists of announcements. CSV or XLS (Excel 97/2003) file may be given as input of prepare.py. Format of the file should be the following:

Code	Date	Class Label	Error Cost	Success Gain	Announcement
MOVIE_SHORT	01.01.99	neg	1.5	1	/Users/alexice/nltk_data/corpora/movie_reviews/neg/cv000_29416.txt
MOVIE_SHORT	02.01.99	neg	1	2	/Users/alexice/nltk_data/corpora/movie_reviews/neg/cv001_19502.txt
MOVIE_SHORT	03.01.99	pos	2	1	/Users/alexice/nltk_data/corpora/movie_reviews/pos/cv002_17424.txt
MOVIE_SHORT	04.01.99	pos	0.7	1	/Users/alexice/nltk_data/corpora/movie_reviews/pos/cv003_12683.txt

field name	meaning
Code	Stock symbol or symbol for of group if it is necessary
Date	Associated date (when announcement was issued)
Class label	Only «pos» or «neg»
Error Cost	Amount of money you would lose in a case of incorrect classification
Success Gain	Amount of money you would gain in a case of correct classification
Announcement	path to announcement (only file protocol is supported)

First row SHOULD contain these particular names in this particular order. If you need any extra data or calculations to be included in XLS it is possible to put them on other XLS sheets. First sheet should have this form. For CSV file only these fields and corresponding data allowed as CSV does not provide devision on sheets.
If there are more than one announcements for certain symbol (code) and date there are two possibilities:

- it is possible to have additional record with the same data except «Announcement» field, which should contain path for other announcement for this symbol and date;
- additional announcement paths may be provided in the same row in next after «Announcement» columns, it is possible to provide as many additional announcements as it is allowed by XLS format. In such case there is no necessity to name these additional columns

Announcement should have a form of path (only file-protocol is supported in prepare.py command, because http: requests may lead to high data-provider server load). This path is used as a key in DB. So it is possible to use same CSV/XLS to change labels or profit/loss later if necessary.

Announcement files should be tokenized, this procedure may be necessary only once for particular announcement because tokenization process is not tunable.

So first command in GFTCT workflow is:

```
python prepare.py import data.xls
```

Where data.xls is XLS file of the described format.
This command do the following:

- checks correctness of data.xls format
- imports every record from data.xls to DB -- on this stage md5 hash of file by path is used, so if there is already same file in DB it will not be imported, if you want to rewrite old record with new one use --modify option
- copies every announcement file to local application directory ./data/files (if you don't want to copy files use --no-copy option, but if your announcements are in pdf format from the site you showed, you should not use this option, because those pdfs are protected from reading and when they are copied by this command protection is also removed, so new pdf file with no reading protection is saved locally)
- if your announcements are in html format, you may want to use only part of all html page code for tokenization (there could be some menu items in page, company copyrights, etc.), so it is possible to provide XPath to necessary element of page with --xpath option. This option have no effect on pdf or text files.

Once correctly imported data need not be reimported further.
As soon as records are imported they should be tokenized. Actually it is possible to make tokenization while importing:

```
python prepare.py import --tokenize data.xls
```

But there is a special separate command of prepare.py for tokenization:

```
python prepare.py tokenize
```

With this command you may also specify symbols (--symbols option) if you want to tokenize only announcements of certain company. Or you may call it with --force option which forces retokenization of all records it may be necessary in a case of error of previous call (for example keybreak or energy switch off).
Tokenization not only tokenize text, but also make token filtration and lemmatization. Only words of form r«\A\w+-?\w+\Z» are accepted it means: words which consist of only letters and maybe one hyphen symbol between these letters. Then words are lemmatized by WordNet lemmatizer and «went» becomes «go», «profits» becomes «profit», etc.

Finally you may use «label» command, but it is not necessary at this stage because you should already have correct labels and profit/loss estimation in data.xls. You may use it later, for more precise estimations of profits / losses working with training/test module.

```
python prepare.py label data-new.xls
```

This call is rather fast, because it makes no any text parsing or file conversion/coping it just updates records in DB.

Some additional notes on prepare.py

If you have used *prepare.py import* with *--no-copy* flag on protected pdfs then you will get Exception on tokenization. To fix it you should reimport corresponding files:

```
python prepare.py import --modify data.xls
```

If you damaged DB somehow it may be better to remove directory ./data with all files and then run

```
python app_setup.py
```

And make all procedure from beginning.

You may use

```
python prepare.py label --xpath //newxpath data-new.xls
```

to change xpath. If you have occasionally provided incorrect xpath while importing. After this you'll have to make:

```
python prepare.py tokenize
```

To retokenize records with html announcements.

It is not necessary to copy files if there are no protected pdfs among input announcement files, you just have to leave them in their places (as it is described in paths), if you remove them then they can not be found when *tokenize* command will be invoked.

prepare.py outputs only aggregated results, information about processing of each record is placed to *prepare.log* file.

Run

```
python prepare.py -h
```

or

```
python prepare.py <command> -h
```

to get help on options.

transform.py

This command is intended to create transformation and vectorize samples.
You have to run it with configuration file:

```
python transform.py transform.cfg
```

Sample configuration file (*smp.cfg*) is provided.
transform.py always outputs transformed (vectorized) samples. It may also output transformation and dictionary in CSV format.
There are two sections in configuration file: «General» and «Transformations and reductions».

General

option	possible values	description
config_id	string	Just string you may use to identify transformation, not necessarily unique
start_date	date in format: MM/DD/YYYY	Earliest date for sample selection (including)
end_date	date in format: MM/DD/YYYY	Latest date for sample selection (including)
symbols	space separated list of symbols or blank	Symbols for sample selection if none of symbols provided then all used
use_transformation_from_file	file name or blank	File with transformation to load if you already have one, if this option provided, then all section «Transformations and reductions» is ignored and transformation from file is used
save_transformation_file_as	file name or blank	File name for this transformation to be saved. You may use this file later in « use_transformation_from_file» option
save_samples_file_as	file name	File name for vectorized samples to be saved

Transformations and reductions

option	possible values	description
feature_selection_subset	float (0..1.0]	Within «General» section it is determined which samples should be transformed, but if you want only random 40% of them to be used for transformation creation you should specify 0.4 here. In that way 40% of samples will be used for transformation creation and will be marked as not to be used as test samples and 60% will be just transformed. All of them will be saved in corresponding file.
consider_only_presence	yes/no	If «yes» then boolean vectors will be created. 1 - if there is at least one word in document, 0 - if there is no such word in document.

option	possible values	description
consider_negations	yes/no	If «yes» then «not good» considered as one word
filter_stopwords	yes/no	If «yes» then english stop-words list is used to filter stop-words
top_words	number	If not 0, then top <number> words occurred in transformation documents are taken for dictionary. Words are sorted in descending order by difference of pos/neg idf (inverse document frequency) and then N=top_words words are taken from the very to («neg» arguments) and N from very bottom of sorted list («pos» arguments).
min_sentences	number	This parameter is used for sample query from database. If you don't want to use very short texts you may set lower limit for number of sentences in the document to be used as sample.
min_token_len	number	Tokens has default minimum length of 2 due to specific of tokenization procedure. But it is possible to filter even longer tokens if necessary.
rare_tokens_threshold	number	Minimum number of documents from transformation set token should occur in to be included into the dictionary.
tfidf	yes/no	Make tfidf transformation (ignored if consider_only_presence=yes)
normalize	yes/no	Make sample vector normalization. Divide vector by its norm. So its new norm becomes equals to 1. (ignored if consider_only_presence=yes)
reduction	yes/no	Use PCA dimension reduction or not.
components	number	Number of PCA components (ignored if reduction=no)

Transformation itself is the object which has:

- a dictionary of fixed length and order
- rules how to calculate numerical values corresponding to each word in dictionary (like put 1 if word occur or put frequency or weight frequency with idf, etc.)
- PCA reduction if necessary

As an input transformation receives text in a form of list of lists (list of sentences were each sentence is represented as list of words (tokens)) or list of texts, transformation outputs vector or matrix.

Samples that was used for transformation creation should not be used as test samples. Because transformation is a kind of implicit training. All sample ids that were used for transformation creation are saved in transformation object so it is possible to filter such samples from test sets (and it is implemented in learn.py) when making testing of learning algorithms.

Ideally it is better to use one set of samples for transformation creation with feature_selection_subset=1. Save transformation obtained (save_transformation_file_as) and then use this transformation (use_transformation_from_file) for making samples for train/test module (learn.py) from new set with no intersections with transformation set.

If you call:

```
python transform.py --help
```

You'll see possible options of transformation.py command. These options allow to override any file option of config file:

option	result
--samples-file SAMPLES_FILE	Override config samples file name to save samples
--transformation-file-save TRANSFORMATION_FILE_SAVE	Override config transformaton file name to save transformation
--transformation-file-load TRANSFORMATION_FILE_LOAD	Override config transformaton file name to load transformation

When calling:

```
python transform.py config.cfg --replace
```

then files with saved samples and transformation and with the same names as in config.cfg will be overwritten. Without this option you'll receive message that samples or transformation file with this name already exists and program will stop.

If you want to obtain csv report with dictionary used by transformation use:

```
python transform.py config.cfg --csv-report report.csv
```

learn.py

This utility makes model training and testing.

```
python learn.py smp.pickle
```

It takes as input argument samples file obtained from transform.py and makes 10 rounds of train-test checks with different subsets of samples. Then it outputs performance metrics for all predictors. And finally it makes training of all predictors by all provided samples and saves prediction models.
By default it doesn't save prediction model - only makes validation, you should use --save-model option to save models:

```
python learn.py smp.pickle --save-model model.pickle
```

This is the example of learn.py console output:

```
RESULTS
```

```
Learner: Support Vector Machines Classifier (linear)
   Total precision: 79.400216 (+/- 7.336398)%
   Precision           : pos:  82.64 (+/- 10.47)% neg:  73.77 (+/-  8.79)%
   Recall              : pos:  64.83 (+/- 12.33)% neg:  87.14 (+/-  6.97)%
   Support             : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed            : pos:   3.70 (+/-  0.67)  neg:   6.00 (+/-  0.77)
   No decision         : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total          : pos:   2.30 (+/-  0.67)  neg:   2.60 (+/-  0.78)
   Gain per supposition: pos:   0.65 (+/-  0.21)  neg:   0.48 (+/-  0.18)
Learner: Nu-Support Vector Machines Classifier (rbf)
   Total precision: 73.114719 (+/- 5.944829)%
   Precision           : pos:  75.00 (+/-  9.98)% neg:  68.86 (+/-  6.10)%
   Recall              : pos:  62.17 (+/-  8.69)% neg:  77.38 (+/-  8.62)%
   Support             : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed            : pos:   4.10 (+/-  0.69)  neg:   5.60 (+/-  0.81)
   No decision         : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total          : pos:   1.70 (+/-  0.67)  neg:   2.00 (+/-  0.55)
   Gain per supposition: pos:   0.50 (+/-  0.20)  neg:   0.38 (+/-  0.12)
Learner: Linear Stochastic Gradient Descent Classifier
   Total precision: 69.531169 (+/- 10.617546)%
   Precision           : pos:  72.64 (+/- 12.31)% neg:  64.27 (+/- 10.99)%
   Recall              : pos:  58.17 (+/- 11.76)% neg:  75.60 (+/- 12.71)%
   Support             : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed            : pos:   4.10 (+/-  0.82)  neg:   5.60 (+/-  0.64)
   No decision         : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total          : pos:   1.30 (+/-  1.16)  neg:   1.60 (+/-  1.00)
   Gain per supposition: pos:   0.45 (+/-  0.25)  neg:   0.29 (+/-  0.22)
Learner: k-nearest neighbors Classifier
   Total precision: 70.071717 (+/- 7.173812)%
   Precision           : pos:  67.56 (+/- 10.19)% neg:  70.50 (+/-  6.10)%
   Recall              : pos:  73.67 (+/-  7.54)% neg:  62.38 (+/- 11.69)%
   Support             : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed            : pos:   5.50 (+/-  0.87)  neg:   4.20 (+/-  0.58)
   No decision         : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total          : pos:   1.50 (+/-  0.87)  neg:   1.80 (+/-  0.66)
   Gain per supposition: pos:   0.35 (+/-  0.20)  neg:   0.41 (+/-  0.12)
Learner: Naive Bayes Classifier (Bernoulli)
   Total precision: 78.418561 (+/- 4.555155)%
   Precision           : pos:  80.17 (+/-  8.88)% neg:  74.33 (+/-  6.99)%
   Recall              : pos:  66.00 (+/- 11.82)% neg:  81.31 (+/-  8.58)%
   Support             : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed            : pos:   4.10 (+/-  0.93)  neg:   5.60 (+/-  1.03)
   No decision         : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total          : pos:   2.10 (+/-  0.61)  neg:   2.40 (+/-  0.51)
   Gain per supposition: pos:   0.60 (+/-  0.18)  neg:   0.49 (+/-  0.14)
Learner: Always 'pos' Classifier [for camparison]
   Total precision: 24.560795 (+/- 3.788702)%
   Precision           : pos:  48.98 (+/-  3.79)% neg:   0.00 (+/-  0.00)%
   Recall              : pos: 100.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Support             : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed            : pos:   9.70 (+/-  0.74)  neg:   0.00 (+/-  0.00)
   No decision         : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total          : pos:  -0.30 (+/-  0.74)  neg:   0.00 (+/-  0.00)
   Gain per supposition: pos:  -0.02 (+/-  0.08)  neg:    nan (+/-   nan)
Learner: Always 'neg' Classifier [for camparison]
   Total precision: 26.606250 (+/- 3.811544)%
   Precision           : pos:   0.00 (+/-  0.00)% neg:  51.02 (+/-  3.79)%
   Recall              : pos:   0.00 (+/-  0.00)% neg: 100.00 (+/-  0.00)%
   Support            : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed           : pos:   0.00 (+/-  0.00)  neg:   9.70 (+/-  0.74)
   No decision        : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total         : pos:   0.00 (+/-  0.00)  neg:   0.30 (+/-  0.74)
   Gain per supposition: pos:    nan (+/-   nan)  neg:   0.02 (+/-  0.08)
Learner: Rangom Classifier [for camparison]
   Total precision: 55.767100 (+/- 8.315872)%
   Precision          : pos:  55.08 (+/- 10.20)% neg:  52.29 (+/- 12.10)%
   Recall             : pos:  57.50 (+/- 11.99)% neg:  49.29 (+/- 13.09)%
   Support            : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed           : pos:   5.40 (+/-  1.14)  neg:   4.30 (+/-  0.81)
   No decision        : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total         : pos:   0.20 (+/-  0.77)  neg:   0.50 (+/-  0.72)
   Gain per supposition: pos:   0.10 (+/-  0.20)  neg:   0.05 (+/-  0.24)
Learner: Agregated (SVC, nuSVC, SGDC, KNeighborsC, BernoulliNB) [majority]
   Total precision: 79.299567 (+/- 5.398098)%
   Precision          : pos:  81.64 (+/-  8.62)% neg:  75.02 (+/-  6.86)%
   Recall             : pos:  68.83 (+/- 10.36)% neg:  82.98 (+/-  7.34)%
   Support            : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed           : pos:   4.10 (+/-  0.72)  neg:   5.60 (+/-  0.84)
   No decision        : pos:   0.00 (+/-  0.00)% neg:   0.00 (+/-  0.00)%
   Gain total         : pos:   2.30 (+/-  0.55)  neg:   2.60 (+/-  0.60)
   Gain per supposition: pos:   0.63 (+/-  0.17)  neg:   0.50 (+/-  0.14)
Learner: Agregated (SVC, nuSVC, SGDC, KNeighborsC, BernoulliNB) > 2
   Total precision: 80.720563 (+/- 5.877651)%
   Precision          : pos:  85.50 (+/-  8.38)% neg:  74.71 (+/-  7.12)%
   Recall             : pos:  62.33 (+/- 11.06)% neg:  75.95 (+/-  9.66)%
   Support            : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed           : pos:   3.50 (+/-  0.64)  neg:   5.10 (+/-  0.88)
   No decision        : pos:   8.50 (+/-  6.90)% neg:  12.86 (+/-  6.71)%
   Gain total         : pos:   2.30 (+/-  0.59)  neg:   2.30 (+/-  0.59)
   Gain per supposition: pos:   0.71 (+/-  0.17)  neg:   0.49 (+/-  0.14)
Learner: Agregated (SVC, nuSVC, SGDC, KNeighborsC, BernoulliNB) [all]
   Total precision: 73.480303 (+/- 13.121632)%
   Precision          : pos:  87.50 (+/- 15.05)% neg:  58.00 (+/- 17.37)%
   Recall             : pos:  38.00 (+/- 12.95)% neg:  40.60 (+/- 15.96)%
   Support            : pos:   4.70 (+/-  0.39)  neg:   5.00 (+/-  0.63)
   Supposed           : pos:   2.10 (+/-  0.57)  neg:   2.50 (+/-  0.78)
   No decision        : pos:  46.83 (+/- 11.01)% neg:  54.88 (+/- 14.33)%
   Gain total         : pos:   1.50 (+/-  0.75)  neg:   1.10 (+/-  0.61)
   Gain per supposition: pos:   0.75 (+/-  0.30)  neg:    nan (+/-   nan)
```

First five learners are scikit-implemented ML algorithms:

- Support Vector Machines Classifier (linear)
- Nu-Support Vector Machines Classifier (rbf)
- Linear Stochastic Gradient Descent Classifier
- k-nearest neighbors Classifier
- Naive Bayes Classifier (Bernoulli)

Next three classifiers are not classifiers at all, they are just for comparison and better understanding of metrics:

- Always 'pos' Classifier
- Always 'neg' Classifier
- Random Classifier

You can easily guess from their names how they actually «solve» classification task :-)
Last three classifiers are just ensembles of first fives:

- Agregated (SVC, nuSVC, SGDC, KNeighborsC, BernoulliNB) [majority]
- Agregated (SVC, nuSVC, SGDC, KNeighborsC, BernoulliNB) > 2
- Agregated (SVC, nuSVC, SGDC, KNeighborsC, BernoulliNB) [all]

First of them makes classification by prevailing opinion, since there are five (odd number) primary classifiers it always give you answer: «pos» or «neg». Second gives you answer when only one or less has opinion against others (so it capable of giving you no answer). Last gives you an answer when all classifiers agree with each other.

Now about metrics:

Precision

name	meaning
precision	$tp/(tp+fp)$, tp - true positives fp - false positives The ability of classifier not to label as positive a sample that is negative. But in these terms positives and negatives are not our «pos» and «neg», actually they are for «pos» precision metric, but for «neg» precision metric it should be reformulated: the ability of classifier not to label as «neg» a sample that is «pos». Marginal cases: if you have 100 samples with 50/50 pos/neg and 1. your classifier makes only one and correct guesses of «pos» and all other labels as «neg», it will have 100% «pos» precision and 50.50% «neg» precision 2. your classifier makes random 50/50 classification, it will have ~50% for both «pos» and «neg» precision 3. ideal classifier will have both «pos» and «neg» precision 100%
total precision	The ability of classifier not to make a mistake
recall	$tp/(tp+fn)$ fn - false negatives The ability of classifier to find all samples of a given class. Marginal cases: if you have 100 samples with 50/50 pos/neg and 1. your classifier makes only one and correct guesses of «pos» and all other labels as «neg», it will have 2% «pos» recall and 100% «neg» recall 2. always «pos» classifier will have 100% «pos» recall and 0% «neg» recall 3. ideal classifier will have both «pos» and «neg» recall 100%
support	Number of samples of a given class used in experiment. As you can see from log it can be real number because all metrics including this one are averaged by 10 experiments, so it just means that there were different amounts of test samples in experiments
supposed	Number of suppositions classifier made for a given class
no decision	This metric concerns to last two classifiers only, because they may give you no decision. It is a percentage of samples that were not classified when actually sample had a given class.

name	meaning
gain total	Sum of gains for correct prediction minus sum of losses for incorrect predictions for a given class. This metric allows you to estimate economical efficiency of classifier, but gains and losses should be correctly calculated and provided to prepare.py with label subcommand
gain per supposition	Gain (or loss if with minus) per supposition (decision). If there are not very many samples for experiment you may obtain nan values for this metric it is because in at least one of the 10 experiment there were no suppositions of given class so there is devision by zero.

Every metric is output with +/- std/2 (half of standard deviation) so that you may see most probable ranges of real value. The more samples are used for tests the less these tolerances but the more time is necessary for calculations.

About weights:
It is possible to use ether losses («Error loss» in XLS/CSV for prepare.py) or gains («Success gain») as weights for samples. So samples with higher weights are considered to be more important by classifier when it learns them.
Run

```
python learn.py smp.pickle --use-loss
```

or

```
python learn.py smp.pickle --use-gain
```

It should be mentioned that you may relabel samples or change «Error loss» or «Success gain» even after sample generation with *prepare.py label*. Because labels and gains/losses are taken from database by learn.py not from samples file.

If you don't want to make validation, just train and save model for further using with predict.py you may call so:

```
python learn.py smp.pickle --no-test --save-model model.pickle
```

NB: Don't remove or rename corresponding transformation file if you want to use your model for prediction! Model without corresponding transformation is useless. Prediction model and transformation are saved in separate files.

predict.py
predict.py is used for prediction:

```
python predict.py  model.pickle text_to_classify.txt
```

The output of predict.py is like this:

```
Support Vector Machines Classifier (linear)                 : [ pos ]
Nu-Support Vector Machines Classifier (rbf)                 : [ pos ]
Linear Stochastic Gradient Descent Classifier               : [ pos ]
k-nearest neighbors Classifier                              : [ pos ]
Naive Bayes Classifier (Bernoulli)                          : [ neg ]
```

You may use txt, html or pdf files as input to classify.
If you call

```
python predict.py  model.pickle text_to_classify.txt --report
```

then utility will not only output predictions but also creates report.csv (overwrites if file exists) with words that are both in transformation dictionary and sample file with counts idfs, etc.
In most cases (when you don't use reduction in transformation) report.csv agrees with Naive Bayes Classifier (Bernoulli) opinion, because this classifier works exactly by counting probabilities based on vector component values asymmetry for different classes. But if you use reduction then Naive Bayes Classifier (Bernoulli) works with vector components which have different meaning than just word counts, anyway even in this case report.csv with words may be generated and analyzed.

One more utility for convenience (convert.py)
This tool is for conversion pdfs to txt.
Some pdfs may be password protected so it could take rather long time to convert them. If you use prepare.py built-in conversion and there will be some error or interruption during process it could be necessary to repeat all from beginning. It is better to convert all pdfs in advance and then just use --no-copy option when importing with prepare.py.

DB Inspection
If you want to inspect data in DB you just need any sqlite browsing utility, this one for example:
http://sqlitebrowser.sourceforge.net/
Default DB path is: ./data/data.sqlite
NB: You can not see in normal form content of some fields (tokenized text for instance) because it is represented as pickle object in DB. You may access these fields by SQLObject API programmatically. Actually it is rather easy in Python:

```
from app_libs.db_proc import *
ID = 1
an = Announcement.get(ID)
print an.tokenization.text_tokenized
```

You should run this code from application directory and some announcements should be imported and tokenazed .

Some additional notes

Note about particular words value for classification

Let's imagine hypothetical situation when we have to classify documents as «pos» or «neg» and the implicit rule (we don't know it but it exists) is the following:

1. if word[1] occur and (word[2] and word[3] don't occur) then class is «pos»
2. if word[1] don't occur and (word[2] and word[3] occur) then class is «pos»
3. else class is «neg»

All other words may help determine class with some probabilities, but words 1..3 allow to determine class exactly. And we have constructed matrix:

	pos	pos	pos	neg	neg	neg
word 1	1	0	1	1	0	1
word 2	0	1	0	1	0	0
word 3	0	1	0	0	0	1
word 4	1	1	0	0	0	1
	doc 1	**doc 2**	**doc 3**	**doc 4**	**doc 5**	**doc 6**

As one may see, for each word 1..3 conditional frequencies for «pos» and «neg» are equal, for word 4 they are not. But if we had used word ranking and dictionary cut (using top_words option) we would have lost words 1..3 because their pos_idf = neg_idf.
So it is obvious that there could be situations when presence or absence of some words are not valuable separately but only with some conditions on other words presence or absence. And whether it is so or not can not be determined in advance and is subject area specific.