django-recommends Documentation

Release 0.4.0

Flavio Curella

November 28, 2016

Contents

1 Quickstart 3

2 Recommendation Providers 5
- 2.1 Properties . . . 7
- 2.2 Methods . . . 7

3 Recommendation Algorithms 9
- 3.1 NaiveAlgorithm . . . 10
- 3.2 RecSysAlgorithm . . . 10

4 Models 11

5 Storage backend 13
- 5.1 RedisStorage . . . 14
- 5.2 DjangoOrmStorage . . . 15
- 5.3 MongoStorage . . . 16

6 Signals 17

7 Template Tags & Filters 19
- 7.1 Filters . . . 19
- 7.2 Tags . . . 19
- 7.3 Templatetags Cache . . . 19

8 Settings 21
- 8.1 Autodiscovery . . . 21
- 8.2 Celery Task . . . 21
- 8.3 Template tags and filters cache timeout . . . 21
- 8.4 Storage backend . . . 21
- 8.5 Logging . . . 22

9 Large Datasets 23

10 Changelog 25

11 Requirements 29
- 11.1 Optional . . . 29

12 Indices and tables 31

A django app that builds item-based suggestions for users.

Contents:

CHAPTER 1

Quickstart

1. Install `django-recommends` with:

```
$ pip install django-recommends
```

2. Create a RecommendationProvider for your models, and register it in your `AppConfig` (see Recommendation Providers)
3. Add `'recommends'` and `'recommends.storages.djangoorm'` to `INSTALLED_APPS`
4. Run `syncdb`

CHAPTER 2

Recommendation Providers

In order to compute and retrieve similarities and recommendations, you must create a `RecommendationProvider` and register it with the model that represents the rating and a list of the models that will receive the votes.

A `RecommendationProvider` is a class that specifies how to retrieve various informations (items, users, votes) necessary for computing recommendation and similarities for a set of objects.

Subclasses override properties amd methods in order to determine what constitutes rated items, a rating, its score, and user.

The algorithm to use for computing is specified by the `algorithm` property.

A basic algorithm class is provided for convenience at `recommends.algorithms.naive.NaiveAlgorithm`, but users can implement their own solutions. See Recommendation Algorithms.

Example:

```
# models.py
from __future__ import unicode_literals
from django.db import models
from django.contrib.auth.models import User
from django.contrib.sites.models import Site
from django.utils.encoding import python_2_unicode_compatible


@python_2_unicode_compatible
class Product(models.Model):
    """A generic Product"""
    name = models.CharField(blank=True, max_length=100)
    sites = models.ManyToManyField(Site)

    def __str__(self):
        return self.name

    @models.permalink
    def get_absolute_url(self):
        return ('product_detail', [self.id])

    def sites_str(self):
        return ', '.join([s.name for s in self.sites.all()])
    sites_str.short_description = 'sites'


@python_2_unicode_compatible
class Vote(models.Model):
```

```
    """A Vote on a Product"""
    user = models.ForeignKey(User, related_name='votes')
    product = models.ForeignKey(Product)
    site = models.ForeignKey(Site)
    score = models.FloatField()

    def __str__(self):
        return "Vote"
```

Create a file called `recommendations.py` inside your app:

```
# recommendations.py

from django.contrib.auth.models import User
from recommends.providers import RecommendationProvider
from recommends.providers import recommendation_registry

from .models import Product, Vote

class ProductRecommendationProvider(RecommendationProvider):
    def get_users(self):
        return User.objects.filter(is_active=True, votes__isnull=False).distinct()

    def get_items(self):
        return Product.objects.all()

    def get_ratings(self, obj):
        return Vote.objects.filter(product=obj)

    def get_rating_score(self, rating):
        return rating.score

    def get_rating_site(self, rating):
        return rating.site

    def get_rating_user(self, rating):
        return rating.user

    def get_rating_item(self, rating):
        return rating.product

recommendation_registry.register(Vote, [Product], ProductRecommendationProvider)
```

All files called `recommendations.py` will be autodiscovered and loaded by `django-recommends`. You can change the default module name, or disable autodiscovery by tweaking the `RECOMMENDS_AUTODISCOVER_MODULE` setting (see Settings), or you could manually import your module in your app's `AppConfig.ready`:

```
# apps.py

from django.apps import AppConfig


class MyAppConfig(AppConfig):
    name = 'my_app'

    def ready(self):
        from .myrecs import *
```

2.1 Properties

- `signals`

  This property define to which signals the provider should listen to. A method of the same name will be called on the provider when the corresponding signal is fired from one of the rated model.

  See Signals.

  Defaults to `['django.db.models.pre_delete']`

- `algorithm`

  Defaults to `recommends.algorithms.naive.NaiveAlgorithm`

2.2 Methods

- `get_items(self)`

  This method must return items that have been voted.

- `items_ignored(self)`

  Returns user ignored items. User can delete items from the list of recommended.

  See recommends.converters.IdentifierManager.get_identifier for help.

- `get_ratings(self, obj)`

  Returns all ratings for given item.

- `get_rating_user(self, rating)`

  Returns the user who performed the rating.

- `get_rating_score(self, rating)`

  Returns the score of the rating.

- `get_rating_item(self, rating)`

  Returns the rated object.

- `get_rating_site(self, rating)`

  Returns the site of the rating. Can be a `Site` object or its ID.

  Defaults to `settings.SITE_ID`.

- `is_rating_active(self, rating)`

  Returns if the rating is active.

- `pre_store_similarities(self, itemMatch)`

  Optional. This method will get called right before passing the similarities to the storage.

  For example, you can override this method to do some stats or visualize the data.

- `pre_delete(self, sender, instance, **kwargs)`

  This function gets called when a signal in `self.rate_signals` is fired from one of the rated models.

  Overriding this method is optional. The default method removes the suggestions for the deleted objected.

See Signals.

CHAPTER 3

Recommendation Algorithms

A Recommendation Algorithm is a subclass of `recommends.algorithms.base.BaseAlgorithm` that implements methods for calculating similarities and recommendations.

Subclasses must implement this methods:

- `calculate_similarities(self, vote_list)`

  Must return an dict of similarities for every object:

  Accepts a list of votes with the following schema:

```
[
    ("<user1>", "<object_identifier1>", <score>),
    ("<user1>", "<object_identifier2>", <score>),
]
```

  Output must be a dictionary with the following schema:

```
[
    ("<object_identifier1>", [
        (<related_object_identifier2>, <score>),
        (<related_object_identifier3>, <score>),
    ]),
    ("<object_identifier2>", [
        (<related_object_identifier2>, <score>),
        (<related_object_identifier3>, <score>),
    ]),
]
```

- `calculate_recommendations(self, vote_list, itemMatch)`

  Returns a list of recommendations:

```
[
    (<user1>, [
        ("<object_identifier1>", <score>),
        ("<object_identifier2>", <score>),
    ]),
    (<user2>, [
        ("<object_identifier1>", <score>),
        ("<object_identifier2>", <score>),
    ]),
]
```

3.1 NaiveAlgorithm

This class implement a basic algorithm (adapted from: Segaran, T: Programming Collective Intelligence) that doesn't require any dependency at the expenses of performances.

3.1.1 Properties

- `similarity`

  A callable that determines the similiarity between two elements.

  Functions for Euclidean Distance and Pearson Correlation are provided for convenience at `recommends.similarities.sim_distance` and `recommends.similarities.sim_pearson`.

  Defaults to `recommends.similarities.sim_distance`

3.2 RecSysAlgorithm

This class implement a SVD algorithm. Requires `python-recsys` (available at https://github.com/ocelma/python-recsys).

`python-recsys` in turn requires `SciPy`, `NumPy`, and other python libraries.

CHAPTER 4

Models

`Recommends` uses these classes to represent similarities and recommendations. These classes don't have be Django Models (ie: tied to a table in a database). All they have to do is implementing the properties descripted below.

class `Similarity`

A `Similarity` is an object with the fellowing properties:

`object`
The source object.

`related_object`
The target object

`score`
How much the `related_object` is similar to `object`.

class `Recommendation`

A `Recommendation` is an object with the fellowing properties:

`object`
The object being suggested to the user.

`user`
The user we are suggesting `object` to.

`score`
How much the `user` is supposed to like `object`.

CHAPTER 5

Storage backend

Results of the computation are stored according to the storage backend defined in `RECOMMENDS_STORAGE_BACKEND` (default to `'recommends.storages.djangoorm.storage.DjangoOrmStorage'`). A storage backend defines how de/serialize and store/retrieve objects and results.

A storage backend can be any class extending `recommends.storages.base.RecommendationStorage` that implements the following methods and properties:

`get_identifier`(*self*, *obj*, **args*, ***kwargs*)
Given an object and optional parameters, returns a string identifying the object uniquely.

`resolve_identifier`(*self*, *identifier*)
This method is the opposite of `get_identifier`. It resolve the object's identifier to an actual model.

`get_similarities_for_object`(*self*, *obj*, *limit*, *raw_id=False*)

if raw_id = False: Returns a list of `Similarity` objects for given `obj`, ordered by score.

else: Returns a list of similar `model` ids[pk] for given `obj`, ordered by score.

Example:

```
[
    {
        "related_object_id": XX, "content_type_id": XX
    },
    ..
]
```

`get_recommendations_for_user`(*self*, *user*, *limit*, *raw_id=False*)

if raw_id = False: Returns a list of Recommendation objects for given `user`, order by score.

else: Returns a list of recommended `model` ids[pk] for given `user`, ordered by score.

Example:

```
[
    {
        "object_id": XX, "content_type_id": XX
    },
    ..
]
```

`get_votes`(*self*)
Optional.

Retrieves the vote matrix saved by `store_votes`.

You won't usually need to implement this method, because you want to use fresh data. But it might be useful if you want some kind of heavy caching, maybe for testing purposes.

store_similarities(*self*, *itemMatch*)

store_recommendations(*self*, *user*, *recommendations*)
Stores all the recommendations.

`recommendations` is an iterable with the following schema:

```
(
    (
        <user>,
        (
            (<object_identifier>, <score>),
            (<object_identifier>, <score>)
        ),
    )
)
```

store_votes(*self*, *iterable*)
Optional.

Saves the vote matrix.

You won't usually need to implement this method, because you want to use fresh data. But it might be useful if you want to dump the votes on somewhere, maybe for testing purposes.

`iterable` is the vote matrix, expressed as a list of tuples with the following schema:

```
[
    ("<user_id1>", "<object_identifier1>", <score>),
    ("<user_id1>", "<object_identifier2>", <score>),
    ("<user_id2>", "<object_identifier1>", <score>),
    ("<user_id2>", "<object_identifier2>", <score>),
]
```

remove_recommendations(*self*, *obj*)
Deletes all recommendations for object `obj`.

remove_similarities(*self*, *obj*)
Deletes all similarities that have object `obj` as source or target.

get_lock(*self*)
Optional. Acquires an exclusive lock on the storage is acquired. Returns `True` if the lock is aquired, or `False` if the lock is already acquired by a previous process.

release_lock(*self*)
Optional. Releases the lock acquired with the `get_lock` method.

5.1 RedisStorage

This storage allows you to store results in Redis. This is the recommended storage backend, but it is not the default because it requires you to install redis-server.

5.1.1 Options

`threshold_similarities` Defaults to `0`. Only similarities with score greater than `threshold similarities` will be persisted.

`threshold_recommendations` Defaults to `0`. Only recommendations with score greater than `threshold similarities` will be persisted.

5.1.2 Settings

`RECOMMENDS_STORAGE_REDIS_DATABASE`: A dictionary representing how to connect to the redis server. Defaults to:

```
{
    'HOST': 'localhost',
    'PORT': 6379,
    'NAME': 0
}
```

5.2 DjangoOrmStorage

This is the default storage. It requires minimal installation, but it's also the less performant.

This storage allows you to store results in a database specified by your `DATABASES` setting.

In order to use this storage, you'll also need to add `'recommends.storages.djangoorm'` to your `INSTALLED_APPS`.

5.2.1 Options

`threshold_similarities` Defaults to `0`. Only similarities with score greater than `threshold similarities` will be persisted.

`threshold_recommendations` Defaults to `0`. Only recommendations with score greater than `threshold similarities` will be persisted.

5.2.2 Settings

To minimize disk I/O from the database, Similiarities and Suggestions will be committed in batches. The `RECOMMENDS_STORAGE_COMMIT_THRESHOLD` setting set how many record should be committed in each batch. Defaults to `1000`.

`RECOMMENDS_STORAGE_DATABASE_ALIAS` is used as the database where similarities and suggestions will be stored. Note that you will have to add `recommends.storages.djangoorm.routers.RecommendsRouter` to your settings' `DATABASE_ROUTERS` if you want to use something else than the default database. Default value is set to `'recommends'`.

5.3 MongoStorage

5.3.1 Options

`threshold_similarities` Defaults to `0`. Only similarities with score greater than `threshold similarities` will be persisted.

`threshold_recommendations` Defaults to `0`. Only recommendations with score greater than `threshold similarities` will be persisted.

5.3.2 Settings

`RECOMMENDS_STORAGE_MONGODB_DATABASE`: A dictionary representing how to connect to the mongodb server. Defaults to:

```
{
    'HOST': 'localhost',
    'PORT': 27017,
    'NAME': 'recommends'
}
```

`RECOMMENDS_STORAGE_MONGODB_FSYNC`: Boolean specifying if MongoDB should force writes to the disk. Default to `False`.

CHAPTER 6

Signals

When a signal specified in the provider is fired up by the one of the rated models, Django-recommends automaticaly calls a function with the same name.

You can override this function or connect to a different set of signals on the provider using the *signals* property:

```
from django.db.models.signals import post_save, post_delete

class MyProvider(DjangoRecommendationProvider):
    signals = ['django.db.models.post_save', 'django.db.models.pre_delete']

    def post_save(self, sender, instance, **kwargs):
        # Code that handles what should happen...

    def pre_delete(self, sender, instance, **kwargs):
        # Code that handles what should happen...
```

By default, a `RecommendationProvider` registers a function with the `pre_delete` signal that removes the suggestion for the deleted rated object (via its storage's `remove_recommendation` and `remove_similarity` methods).

CHAPTER 7

Template Tags & Filters

To use the included template tags and filters, load the library in your templates by using `{% load recommends %}`.

7.1 Filters

The available filters are:

`similar:<limit>`: returns a list of Similarity objects, representing how much an object is similar to the given one. The `limit` argument is optional and defaults to `5`:

```
{% for similarity in myobj|similar:5 %}
    {{ similarity.related_object }}
{% endfor %}
```

7.2 Tags

The available tags are:

`{% suggested as <varname> [limit <limit>] %}`: Returns a list of Recommendation (suggestions of objects) for the current user. `limit` is optional and defaults to `5`:

```
{% suggested as suggestions [limit 5]  %}
{% for suggested in suggestions %}
    {{ suggested.object }}
{% endfor %}
```

7.3 Templatetags Cache

By default, the templatetags provided by django-recommends will cache their result for 60 seconds. This time can be overridden via the `RECOMMENDS_CACHE_TEMPLATETAGS_TIMEOUT`.

CHAPTER 8

Settings

8.1 Autodiscovery

By default, `django-recommends` will import and load any modules called `recommendations` within your apps.

You can change the default module name by setting `RECOMMENDS_AUTODISCOVER_MODULE` to the name that you want, or you can disable this behavior by setting it to `False`.

8.2 Celery Task

Computations are done by a scheduled celery task.

The task is run every 24 hours by default, but can be overridden by the `RECOMMENDS_TASK_CRONTAB` setting:

```
RECOMMENDS_TASK_CRONTAB = {'hour': '*/24'}
```

`RECOMMENDS_TASK_CRONTAB` must be a dictionary of kwargs acceptable by celery.schedulers.crontab.

If you don't want to run this task (maybe because you want to write your own), set `RECOMMENDS_TASK_RUN = False`

Additionally, you can specify an expiration time for the task by using the `RECOMMENDS_TASK_EXPIRES` settings, which defaults to `None`.

8.3 Template tags and filters cache timeout

RECOMMENDS_CACHE_TEMPLATETAGS_TIMEOUT controls how long template tags and fitlers cache their results. Default is 60 seconds.

8.4 Storage backend

`RECOMMENDS_STORAGE_BACKEND` specifies which Storage backend class to use for storing similarity and recommendations. Defaults to `'recommends.storages.djangoorm.DjangoOrmStorage'`. Providers can override this settings using the `storage` property (see Recommendation Providers).

8.5 Logging

`RECOMMENDS_LOGGER_NAME` specifies which logger to use. Defaults to `'recommends'`.

CHAPTER 9

Large Datasets

Calculating item similarities is computationally heavy, in terms of cpu cycles, amount of RAM and database load.

Some strategy you can use to mitigate it includes:

- Parallelize the precomputing task. This could be achieved by disabling the default task (via `RECOMMENDS_TASK_RUN = False`) and breaking it down to smaller tasks (one per app, or one per model), which will be distributed to different machines using dedicated celery queues.

CHAPTER 10

Changelog

- **v0.4.0**
  - Drop support for Django 1.7.
  - Add support for Django 1.10.
- **v0.3.11**
  - Start deprecating `GhettoAlgorithm` in favor of `NaiveAlgorithm`.
- **v0.3.1**
  - Fix wrong import
- **v0.3.0**
  - Added support for Django 1.9.
- **v0.2.2**
  - Added Python 3.3 Trove classifier to *setup.py*.
- **v0.2.1**
  - Added Python 3.4 Trove classifier to *setup.py*.
- **v0.2.0**
  - Added support for Python 3.4
  - Dropped support for Celery 2.x
- **v0.1.0**
  - Django 1.8 compatibility. Removed support for Django 1.6.
  - Added Providers autodiscovery.
- **v0.0.22**
  - Django 1.7 compatibility. Thanks Ilya Baryshev.
- **v0.0.21**
  - Release lock even if an exception is raised.
- **v0.0.20**
  - Removed lock expiration in Redis Storage.
- **v0.0.19**

  - added storages locking. Thanks Kirill Zaitsev.

- **v0.0.16**
  - renamed `--verbose` option to `--verbosity`.
  - The `recommends_precompute` method is available even with `RECOMMENDS_TASK_RUN = False`.
- **v0.0.15**
  - added `--verbose` option to `recommends_precompute` command.
- **v0.0.14**
  - more verbose `recommends_precompute` command. Thanks WANG GAOXIANG.
  - Introduced ‘‘raw_id‘ parameter for lighter queries. WANG GAOXIANG.
  - Introduced `RECOMMENDS_STORAGE_MONGODB_FSYNC` setting.
- **v0.0.13**
  - Use `{}` instead of `dict()` for better performance.
- **v0.0.12**
  - python 3.3 and Django 1.5 compatibility
- **v0.0.11**
  - `get_rating_site` provider method now defaults to `settings.SITE_ID` instead of `None`.
  - `similarities` templatetag result is now cached per object
  - fixed tests if `recommends_precompute` is None.
  - explicitly named celery tasks.
- **v0.0.10**
  - Added `RecSysAlgorithm`.
- **v0.0.9**
  - Now tests can run in app’s ./manage.py test. Thanks Andrii Kostenko.
  - Added support for ignored user recommendation. Thanks Maxim Gurets.
- **v0.0.8**
  - Added `threshold_similarities` and `threshold_recommnedations` to the storage backends.
- **v0.0.7**
  - added Mongodb storage
  - added Redis storage
  - added `unregister` method to the registry
- **v0.0.6**
  - added logging
  - DjangoOrmStorage now saves Similarities and Suggestions in batches, according to the new `RECOMMENDS_STORAGE_COMMIT_THRESHOLD` setting.
  - Decoupled Algorithms from Providers

- **v0.0.5**
  - Refactored providers registry
  - Renamed recommends.storages.django to recommends.storages.djangoorm to avoid name conflicts
  - Refactored DjangoOrmStorage and moved it to recommends.storages.djangoorm.storage
  - Added optional database router
- **v0.0.4**
  - Refactored providers to use lists of votes instead of dictionaries
  - fixed a critical bug where we ere calling the wrong method with the wrong signature.
- **v0.0.3**
  - Added filelocking to the pre-shipped precomputing task
  - Refactored signal handling, and added a task to remove similarities on pre_delete
  - Added optional hooks for storing and retrieving the vote matrix
- **v0.0.2**
  - Added the `RECOMMENDS_TASK_RUN` setting
- **v0.0.1**
  - Initial Release

CHAPTER 11

Requirements

- Python 2.7, Python 3.3+
- Django>=1.8
- celery>=3
- django-celery>=2.3.3

11.1 Optional

- redis
- pymongo
- python-recsys (Python 2.x only)

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

G

get_identifier(), 13
get_lock(), 14
get_recommendations_for_user(), 13
get_similarities_for_object(), 13
get_votes(), 13

O

object (Recommendation attribute), 11
object (Similarity attribute), 11

R

Recommendation (built-in class), 11
related_object (Similarity attribute), 11
release_lock(), 14
remove_recommendations(), 14
remove_similarities(), 14
resolve_identifier(), 13

S

score (Recommendation attribute), 11
score (Similarity attribute), 11
Similarity (built-in class), 11
store_recommendations(), 14
store_similarities(), 14
store_votes(), 14

U

user (Recommendation attribute), 11