PYCON 2016
ROSE CITY
PORTLAND, OREGON
MAY 28TH – JUNE 5TH

AN ARCHITECTURE FOR MACHINE LEARNING IN DJANGO

Web Applications that Learn by Example

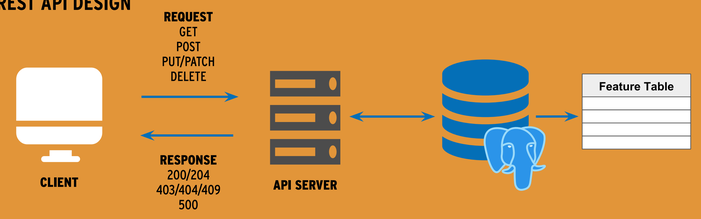
By Benjamin Bengfort and Rebecca Bilbro

DistrictDataLabs
pycon.districtdatalabs.com

How do you operationalize machine learning models in an application?

Web applications are becoming smarter and more personalized through the clever use of automatic optimization and machine learning algorithms. In this poster, we present a standard architecture for Django web applications that utilizes models trained via Scikit-Learn to predict and tune experiences for specific users. Web application developers will discover that the tools they currently use like Django, REST Frameworks, and PostgreSQL are also the framework required for online machine learning. Data scientists will discover that a web platform is ideal for reinforcing models through feedback, employing ensemble techniques, and deciding when to retrain models.

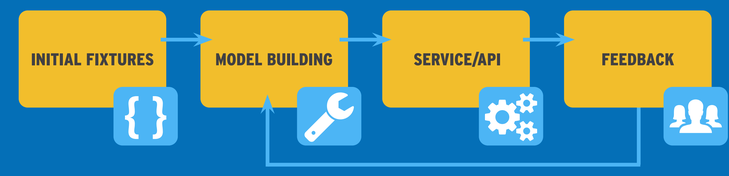
REST API DESIGN



MACHINE LEARNING:
- Batch analysis: learns by iterating over a single table
- Must be routinely updated to take advantage of feedback
- Once trained, is able to make predictions on new data
- Can be evaluated via cross-validation

WEB APPLICATIONS:
- Transactional, deal with only a few rows at a time
- Are live, and perform computation on demand
- Must make immediate responses to requests
- Can be interactive and solicit feedback

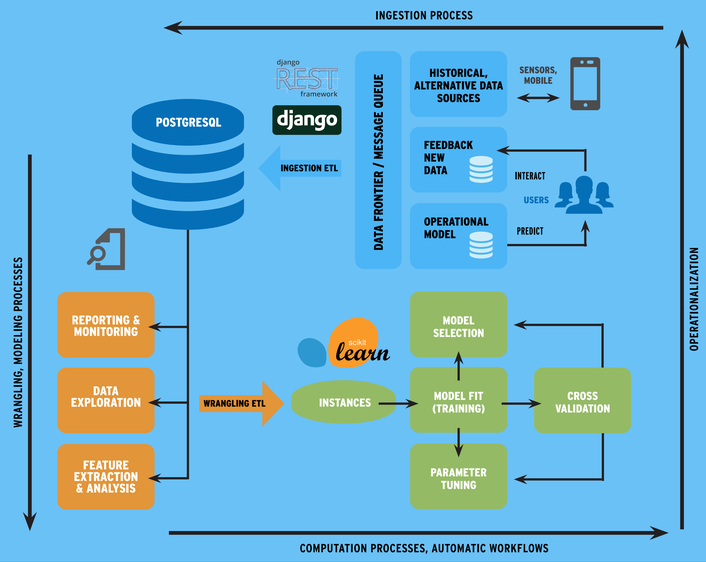
SIMPLE VIEW OF INTEGRATION



Machine learning algorithms differ from traditional data mining in that they utilize pattern recognition techniques from historical data to learn how to identify or predict outcomes for new information. As the algorithm makes predictions, feedback is utilized to adapt or correct the model so that it learns from new experience as well as the old. This makes web and mobile applications an ideal place to employ or deploy machine learned models.

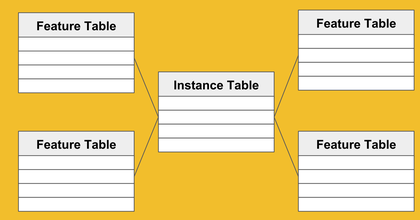
As users interact with your application, they train models that can then be used to tune the experience for new users or to personalize content for existing users.

Machine learning methods have two phases: a training/validation phase and an operational phase. For this reason, traditional web architectures that employ an application server and backend database need to be adapted to fit into the ML lifecycle. In this poster we present a machine learning architecture for directly using models inside of web applications, particularly via a RESTful API.



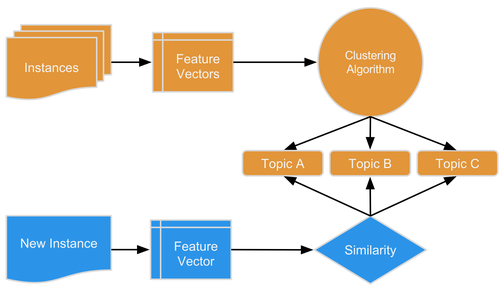
COMPUTATIONAL DATA STORE (ALSO "DATA MANAGEMENT")

Integration happens at the database layer, which we will call a "computational data store". The Web app manages transactions on normalized feature tables, which are joined into an instance table for machine learning.



MACHINE LEARNING PIPELINES

TWO PHASES IN DJANGO



SUPERVISED MACHINE LEARNING – CLUSTERING

OPERATION PHASE

Initialize API by loading "best" model from the database into memory (time consuming, so must be done before request).

Pass GET request to model `predict()`.

Save/update predictions to database and return the predicted response.

Store feedback and update feature tables on POST/PUT/PATCH.

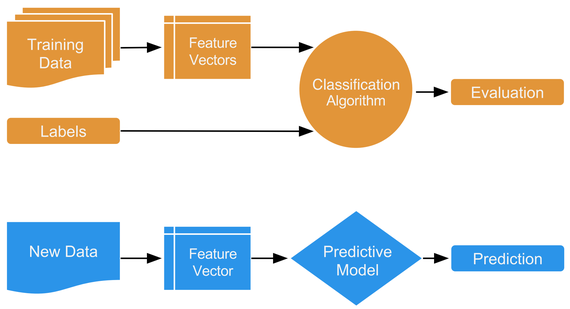
Remove predictions on DELETE.

BUILD PHASE

Routinely (nightly/weekly) join feature tables into an instance table to create a static snapshot of the data to learn on.

Engage the model selection triple to fit one or more models. Evaluate models using cross-validation.

Pickle models and save them back to the database.

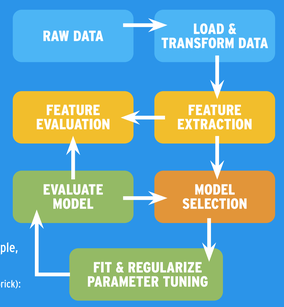


SUPERVISED MACHINE LEARNING – CLASSIFICATION/REGRESSION

MODEL SELECTION TRIPLE

- Feature Analysis
- Hyperparameter Tuning
- Model Selection

Evaluation:
- Visual Evaluation
- Cross-Validation



For more on the model selection triple, check out Yellowbrick (https://github.com/DistrictDataLabs/yellowbrick): a visual diagnostic tool for machine learning.

MODEL STORAGE (ALSO "MODEL MANAGEMENT")

Additionally, models are also stored in the database as pickles, and can be retrieved and loaded by the web application.

ID	Model	Hyperparameters	Build Time	Score	Pickle
1	Naive Bayes	{"alpha": 1.0}	235.32	.832	BLOB
2	SVC	{"C": 1.0, "kernel": "linear"}	20.312	.861	BLOB
3	KNN	{"k": 5, "weights": "distance"}	482.129	.821	BLOB

DATA PRODUCT PIPELINE