Data Science Project Workflow

This blogpost offers an introduction to data science project workflow, some hands-on advice, and links to useful resources.

Prologue: Use the Right Tool

"If the only tool you have is a hammer, you tend to see every problem as a nail." – Abraham Maslow

According to a talk by Anthony Goldbloom, CEO of Kaggle, there are only two winning approaches for all Kaggle competitions:

1. Handcrafted feature engineering
2. Neural networks and deep learning

That's it.

In the first category, it "has almost always been ensembles of decision trees that have won". Random Forest used to be the big winner, but XGBoost has cropped up, winning practically every competition in the structured data category recently.

On the other hand, for any dataset that contains images or speech problems, deep learning is the way to go. And instead of spending their time doing feature engineering, winners spend their time constructing neural networks.

So, the major takeaway from the talk is that **if you have lots of structured data, the handcrafted approach is your best bet, while if you have unusual or unstructured data (images, sequences, etc.) your efforts are best spent on neural networks**.

I've recently conducted a use case proposal of one Kaggle competition, and analyzed (in the appendix) why deep nerual networks, while theoratically being able to approximate any functions, in practice are easliy dwarfed by ensemble tree algorithms in terms of input/output ratio (man it's hard to tune a neural network). That said, stacking deep nets in the final ensemble might still boost performance, as long as the features extracted are not highly correlated with those from ensemble trees.

Workflow

Here is a workflow I'd propose:

1. Divide Forces
2. Data Setup
3. Literature Review

4. Establish an Evaluation Framework
5. Exploratory Data Analysis
6. Preprocessing
7. Feature Engineering
8. Model(s) Tuning
9. Ensemble

Let's look at each of them in more detail.

Divide Forces

1 + 1 is not always equal to 2 when it comes to team collaboration. More often, it is smaller than 2 due to lots of factors such as individual preferences. How to divide force effciently is key to get great results as a team.

According to some Kaggle winners' interviews, forming teams can make them more competitive as "combining features really helps improve performance". At the same time, it helps to specialize, having **different members focus on different stages of the data pipeline**.

Natually, one plausible plan can be:

- Complete data setup
- Review literature
- Establish a evaluation framework as a team
- People with strong statistical background can tackle the EDA task
- After EDA, people with strong scripting (such as Python) skills can then work on data preprocessing (this step is likely to involve SQL depending on data sources)
- People with domain expertise, strong statistical skills, and/or practical experience in working with features can focus on the feature engineering (for deep learning project this step is not always necessary). The output of feature engineering, as suggest by the 1st place Home Depot Product Search Relevance, should ideally be serialized pandas dataframes.
- People with machine learning expertise should handle model tuning and ensmeble parts
- Last but not least, people with strong storytelling & presentation skills can work on communicating data results to senior executives and clients

In traditional machine learning problems, after the preprocessing step, people who work on feature engineering should be **producing new feature sets frequently so that machine learning people can tune and train models continuously**. On the other hand, in deep learning regime most of the time would be spent on net architecture design and model/parameters tuning.

Clearly documented, reproducible codes and data files are crucial.

Data Setup

- **Get the data**: usually for competitions on Kaggle we can get the data effortlessly by downloading from its website. If we are working with clients, getting data might require further efforts, such as accessing their database or cloud storage.
- **Store the data**: either locally or on the cloud, depending on the situation. Always beneficial to have some backup mechanism in place.

Literature Review

This may be the crucial (yet overlooked) part for competing in Kaggle, or doing data science projects in general. To avoid reinventing the wheels and get inspired on how to preprocess, engineer and model the data, it's worth spend 1/10 to 1/5 of the project time just researching how people previously dealt with similar problems/datasets. Some good places to start are:

- No Free Hunch: the official Kaggle blog, artciles under category *Tutorials* and *Winner's Interviews*
- Kaggle competition: Browse through active/completed competitions, and study the similar projects to the one we are working on
- Google is our friend :)

Time spent on literature review is time well spent.

Establish an Evaluation Framework

Having a sound evaluation framework is crucial for all the works that follow: if we use suboptimal metrics or don't have an effective & unbiased cross validaton strategy that could gudie us to tune generalizable models, we are aiming at the wrong target and wasting our time.

Metrics

In terms of the metrics to use, if we are competing on Kaggle we can find it under *Evalution*. If, however, we are working on some real-world data problem where we are free to craft/choose the ruler to measure how good (or bad) our models are, cares should be given about choosing the 'right' metric that makes the most sense for a domain the problem/data at hand. For example, we all know wrongfully classifying a spam email as non-spam does less harm than classifying the non-spam to be spam (imagine missing an important meeting email from your boss…). Similar case for medical diagnosis. In such situations, simple metric such as accuracy is not enough and we might want to consider other metrics, such as precision, recall, F1, ROC AUC score, etc.

A good place to view different metrics and their use cases is sklearn.metrics page.

Cross Validation (CV) Strategies

The key point for machine learning is to build models that can generalize on unseen data, and a good cross validation strategy help us achieve that (while a bad strategy misleads us and gives us false confidence).

For theories on cross validation stategies, here are some good reads:

- 10-fold CV: some discussion regarding why 10 is the magic number
- Nested cross-validation: Varma and Simon concluded that the true error of the estimate is almost unbiased relative to the test set when nested cross-validation is used
- Cross validaton strategy when blending/stacking

Here is one simple example of nested cross-validation in Python using scikit-learn:

```
import numpy as np
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import cross_val_score
from sklearn.ensemble import RandomForestClassifier

X_train = ... # your training features
y_train = ... # your training labels

gs = GridSearchCV(
  estimator = RandomForestClassifier(random_state=0),
  param_grid = {
    'n_estimators': [100, 200, 400, 600, 800],
     # add other params to tune
     }
  scoring = 'accuracy',
  cv = 5
)

scores = cross_val_score(
  gs,
  X_train,
  y_train,
  scoring = 'accuracy',
  cv = 2
)

print 'CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores))
```

The inner loop `GridSearchCV` is used to tune parameters, while the outer loop `cross_val_score` is used to train with optimal parameters and report an unbiased score. This is known as **2x5 cross-validation** (although we can do 3x5, 5x4, or any fold combinations that work for our data size under certain computational constraints).

Exploratory Data Analysis (EDA)

EDA is an approach to analyze data sets and summarize their main characteristics, often with plots. EDA helps data scientists get a better understanding of the dataset at hands, and guide them to preprocess data and engineer features effectively. Some good resources to help carry out effective EDA are:

- Think Stats 2: an introduction to probability and statistics for Python programmers (its github repository holds some useful codes and ipython notebooks for conducting exploratory data analysis)
- Data Science with Python & R: Exploratory Data Analysis
- There are people who usually post their code for EDA on Kaggle forum at the beginning of each competition, or the release of new datasets. Check the forum frequently to get inspired.

Preprocessing

The one and only reason we need to preprocess data is so that a machine learning algorithm can learn most effectively from them. Specifically, three issues need to be addressed:

- **Sources**: we need to integrate data from multiple sources to improve predictability
- **Quality**: incomplete, noisy, inconsistent data
- **Format**: incompatible data fortmat. E.g., generalized linear model requires one-hot encoding of ordinal variables; for natural language processing (NLP) tasks we need to use word embedding (vector) rather than raw text as input.

In practice, tasks in data preprocessing include:

- **Data cleaning**: fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies
- **Data integration**: use multiple data sources and join/merge data
- **Data transformation**: normalize, aggregate, and embed (word)
- **Data reduction**: reduce the volume but produce the same or similar analytical results (e.g., PCA)
- **Data discretization**: replace numerical attributes with nominal/discrete ones

Good places to start are CCSU course page and scikit-learn documentation for data preprocessing in general and how to do it effectively in Python.

Feature Engineering

"Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering." — Andrew Ng

While in deep learning we usually just normalize the data (e.g., such that image pixels have zero mean and unit variance), in traditional machine learning we need handcrafted features to build accurate models. Doing feature engineering is both art and science, and requires iterative experiments and domain knowledge. Feature engineering boils down to feature selection and creation.

Feature selection

scikit-learn offers some great feature selection methods. They can be categoried as one of the following:

- Removing features with low variance
- Univariate feature selection
- Recursive feature elimination
- Selecting from model

Here is one simple example of selecting features using scikit-learn:

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

X_train = ... # your training features
y_train = ... # your training labels

# can be any estimator that has attribute 'feature_importances_' or 'coef_'
model = RandomForestClassifier(random_state=0)

model.fit(X_train, y_train)

fs = SelectFromModel(model, prefit=True)

X_train_new = fs.transform(X_train) # columns selected
```

In this example, features with zero importance (feature_importances_ = 0) will be eliminated.

Some other good post on feature selection are:

- How do I perform feature selection
- Discover Feature Engineering, How to Engineer Features and How to Get Good at It
- Streamlining feature engineering

Feature creation

This is the part where domain knowledge and creativity come in. For example, insurers can create features that help their machine learning model better identify customers with higher risks; similar for geologists who work with geological data.

But here are some general methods that may help you create features to boost model performance:

- Add **zero_count** for each row (especially for sparse matrix)
- Seperate date into year, month, day, weekday or weekend, etc.
- Add percentile change from feature to feature (or other interactions among features)

After adding new handcrafted features, we need to perform another round of feature selection (e.g., using `SelectFromModel` to elimiate non-contributing features). Note that different classifiers might select different features, and it's imoprtant that features selected using a certain classifier are later trained with the **same** classifier. For classifiers that don't have either `feature_importances_` or `coef_` attribute (e.g., nonparametric classifiers such as KNN), the best way is to cross validate the features selected from various classifiers (i.e., to select the set of feature that has the highest CV score).

Feature engineering is the jewel in crown of (traditional) machine learning. As machine learning professor Pedro Domingos puts it: "...some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used."

Model(s) Tuning

If we've got this far, model tuning is actually the easy part: we simply need to provide the parameter search space (starting from coarse, and then to refined ranges) with certain classifiers, and hit the run button and let the machines do all the heavy lifting. The nest CV strategy discussed above will do the job.

If only things were this simple. In practice, however, we face a big constraint: time (deadline to deliver the model to clients or submit to Kaggle competition). To add more complexity, single model never generates the best results, meaning we need to stack many models together to deliver great performance. How to effectively stack (or ensemble) models will be discussed in the next section.

So, one alternative is to optimize the search space automatically, rather than manually setting each parameter from the coarse to the refined. Several Kaggle winners use and recommend hyperopt, a Python library for serial and parallel parameter optimization.

How to effectively use hyperopt may be explained in a later blogpost.

Ensemble

Training one machine learning model (e.g., XGBoost, Random Forest, KNN, or SVM) can give us decent results, but not the best ones. If in practice we want to boost the performance, we might want to stack models using a combiner, such as logistic regression. This 1st place solution for Otto Group Product Classification Challenge is AWESOME.

Combining the winner's ensemble method with other research I've done, I'd love to propose one effective ensembling strategy:

- Preprocessing and feature engineering (discussed above)
- **First layer**: Use nested CV to train N models (1st place of Otto competition used same fold indices), generating N * C (C== num_class) probabilistic predictions (or, meta features) for each data point.
- **Second layer**
    - Single combiner: use the N * C meta features from first layer, plus optionally handcrafted features to train another model (e.g, logistic regression) with nested CV.
    - Multiple combiners: train multiple single combiners with nest CV (1st place of Otto competition used random indices across different combiners).
- **Third layer** (for multiple combiners): cross validate the weights of combiners in the second layer (can be simple linear weights or complex ones such as geometric mean)

From my experience, ensembling can usually boost some % performance, depending on the corrleations of the derivative features. So it's always a good idea to try out.

That's all for this blogpost! Hope it helps.