Approaching (Almost) Any Machine Learning Problem | Abhishek Thakur

Kaggle Team (http://blog.kaggle.com/author/kaggleteam/) | 07.21.2016

29

(http://l

almost-

any-

machin

learning

problen

abhishe

thakur/

Abhishek Thakur (https://www.kaggle.com/abhishek), a Kaggle Grandmaster, originally published this post here (https://www.linkedin.com/pulse/approaching-almost-any-machine-learning-problem-abhishek-thakur) on July 18th, 2016 and kindly gave us permission to cross-post on No Free Hunch

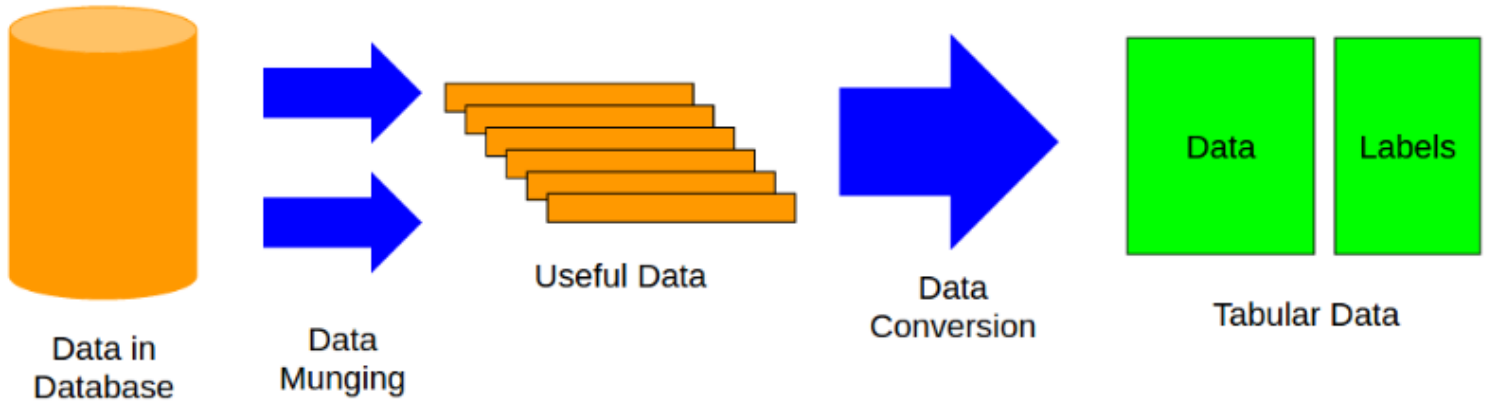
---

An average data scientist deals with loads of data daily. Some say over 60-70% time is spent in data cleaning, munging and bringing data to a suitable format such that machine learning models can be applied on that data. This post focuses on the second part, i.e., applying machine learning models, including the preprocessing steps. The pipelines discussed in this post come as a result of over a hundred machine learning competitions that I've taken part in. It must be noted that the discussion here is very general but very useful and there can also be very complicated methods which exist and are practised by professionals.

We will be using python!

Data

Before applying the machine learning models, the data must be converted to a tabular form. This whole process is the most time consuming and difficult process and is depicted in the figure below.



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_1.png)

The machine learning models are then applied to the tabular data. Tabular data is most common way of representing data in machine learning or data mining. We have a data table, rows with different samples of the data or X and labels, y. The labels can be single column or multi-column, depending on the type of problem. We will denote data by X and labels by y.

Types of labels

The labels define the problem and can be of different types, such as:

- Single column, binary values (classification problem, one sample belongs to one class only and there are only two classes)
- Single column, real values (regression problem, prediction of only one value)
- Multiple column, binary values (classification problem, one sample belongs to one class, but there are more than two classes)
- Multiple column, real values (regression problem, prediction of multiple values)
- And multilabel (classification problem, one sample can belong to several classes)

Evaluation Metrics

For any kind of machine learning problem, we must know how we are going to evaluate our results, or what the evaluation metric or objective is. For example in case of a skewed binary classification problem we generally choose area under the receiver operating characteristic curve (ROC AUC or simply AUC). In case of multi-label or multi-class classification problems, we generally choose categorical cross-entropy or multiclass log loss and mean squared error in case of regression problems.

I won't go into details of the different evaluation metrics as we can have many different types, depending on the problem.

The Libraries

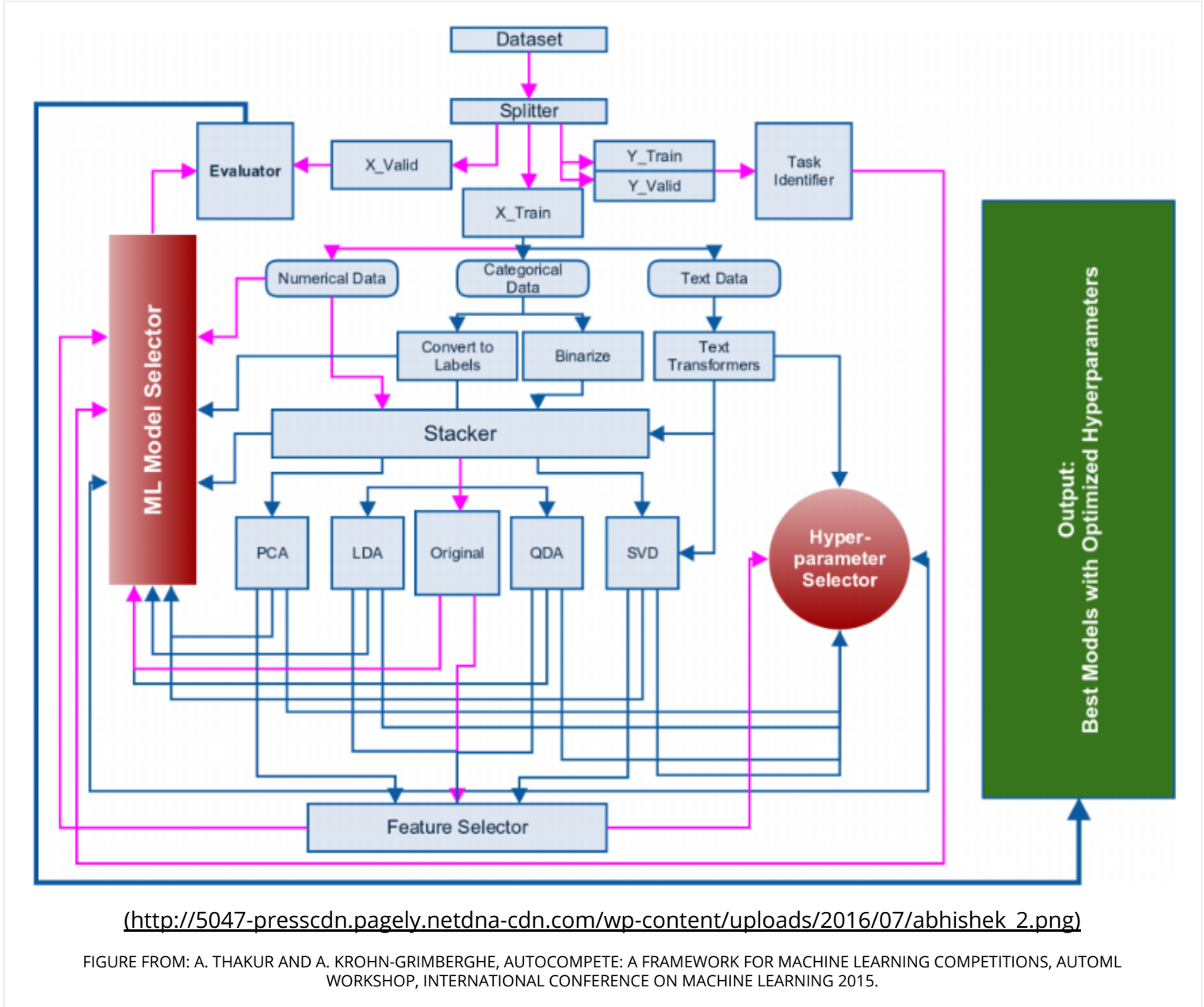
To start with the machine learning libraries, install the basic and most important ones first, for example, numpy and scipy.

- To see and do operations on data: pandas (http://pandas.pydata.org/ (http://pandas.pydata.org/))
- For all kinds of machine learning models: scikit-learn (http://scikit-learn.org/stable/ (http://scikit-learn.org/stable/))
- The best gradient boosting library: xgboost (https://github.com/dmlc/xgboost (https://github.com/dmlc/xgboost))
- For neural networks: keras (http://keras.io/ (http://keras.io/))
- For plotting data: matplotlib (http://matplotlib.org/ (http://matplotlib.org/))
- To monitor progress: tqdm (https://pypi.python.org/pypi/tqdm (https://pypi.python.org/pypi/tqdm))

I don't use Anaconda (https://www.continuum.io/downloads (https://www.continuum.io/downloads)). It's easy and does everything for you, but I want more freedom. The choice is yours. 🙂

The Machine Learning Framework

In 2015, I came up with a framework for automatic machine learning which is still under development and will be released soon. For this post, the same framework will be the basis. The framework is shown in the figure below:

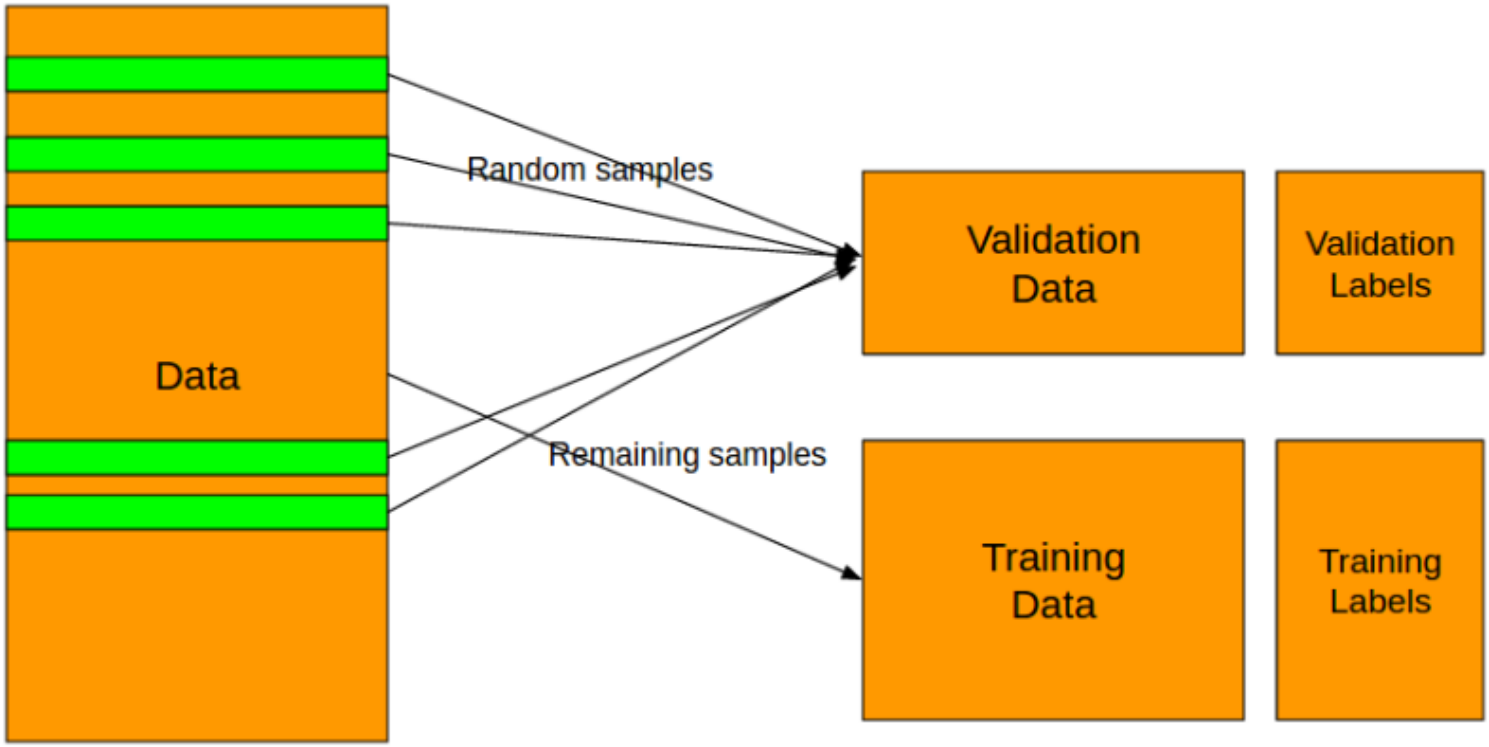


(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_2.png)

FIGURE FROM: A. THAKUR AND A. KROHN-GRIMBERGHE, AUTOCOMPETE: A FRAMEWORK FOR MACHINE LEARNING COMPETITIONS, AUTOML WORKSHOP, INTERNATIONAL CONFERENCE ON MACHINE LEARNING 2015.

In the framework shown above, the pink lines represent the most common paths followed. After we have extracted and reduced the data to a tabular format, we can go ahead with building machine learning models.

The very first step is identification of the problem. This can be done by looking at the labels. One must know if the problem is a binary classification, a multi-class or multi-label classification or a regression problem. After we have identified the problem, we split the data into two different parts, a training set and a validation set as depicted in the figure below.



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_3.png)

The splitting of data into training and validation sets “must” be done according to labels. In case of any kind of classification problem, use stratified splitting. In python, you can do this using scikit-learn very easily.

```
from sklearn.cross_validation import StratifiedKFold
eval_size = 0.10
kf = StratifiedKFold(y, round(1. / eval_size))
train_indices, valid_indices = next(iter(kf))
X_train, y_train = X[train_indices], y[train_indices]
X_valid, y_valid = X[valid_indices], y[valid_indices]
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_4.png)

In case of regression task, a simple K-Fold splitting should suffice. There are, however, some complex methods which tend to keep the distribution of labels same for both training and validation set and this is left as an exercise for the reader.

```
from sklearn.cross_validation import KFold
eval_size = 0.10
kf = KFold(len(y), round(1. / eval_size))
train_indices, valid_indices = next(iter(kf))
X_train, y_train = X[train_indices], y[train_indices]
X_valid, y_valid = X[valid_indices], y[valid_indices]
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_5.png)

I have chosen the eval_size or the size of the validation set as 10% of the full data in the examples above, but one can choose this value according to the size of the data they have.

After the splitting of the data is done, leave this data out and don't touch it. Any operations that are applied on training set must be saved and then applied to the validation set. Validation set, in any case, should not be joined with the training set. Doing so will result in very good evaluation scores and make the user happy but instead he/she will be building a useless model with very high overfitting.

Next step is identification of different variables in the data. There are usually three types of variables we deal with. Namely, numerical variables, categorical variables and variables with text inside them. Let's take example of the popular Titanic dataset (https://www.kaggle.com/c/titanic/data (https://www.kaggle.com/c/titanic/data)).

```
VARIABLE DESCRIPTIONS:
survival        Survival
                (0 = No; 1 = Yes)
pclass          Passenger Class
                (1 = 1st; 2 = 2nd; 3 = 3rd)
name            Name
sex             Sex
age             Age
sibsp           Number of Siblings/Spouses Aboard
parch           Number of Parents/Children Aboard
ticket          Ticket Number
fare            Passenger Fare
cabin           Cabin
embarked        Port of Embarkation
                (C = Cherbourg; Q = Queenstown; S = Southampton)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_6.png)

Here, survival is the label. We have already separated labels from the training data in the previous step. Then, we have pclass, sex, embarked. These variables have different levels and thus they are categorical variables. Variables like age, sibsp, parch, etc are numerical variables. Name is a variable with text data but I don't think it's a useful variable to predict survival.

Separate out the numerical variables first. These variables don't need any kind of processing and thus we can start applying normalization and machine learning models to these variables.

There are two ways in which we can handle categorical data:

- Convert the categorical data to labels

```
from sklearn.preprocessing import LabelEncoder

lbl_enc = LabelEncoder()
lbl_enc.fit(xtrain[categorical_features])
xtrain_cat = lbl_enc.transform(xtrain[categorical_features])
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_7.png)

- Convert the labels to binary variables (one-hot encoding)

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()
ohe.fit(xtrain[categorical_features])
xtrain_cat = ohe.transform(xtrain[categorical_features])
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_8.png)

Please remember to convert categories to numbers first using LabelEncoder before applying OneHotEncoder on it.

Since, the Titanic data doesn't have good example of text variables, let's formulate a general rule on handling text variables. We can combine all the text variables into one and then use some algorithms which work on text data and convert it to numbers.

The text variables can be joined as follows:

```
text_data = list(X_train.apply(lambda x:'%s %s' %(x['column_1'],x['column_2']),axis=1))
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_9.png)

We can then use CountVectorizer or TfidfVectorizer on it:

```
from sklearn.feature_extraction.text import CountVectorizer
ctv = CountVectorizer()
text_data_train = ctv.fit_transform(text_data_train)
text_data_valid = ctv.fit_transform(text_data_valid)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_10.png)

or,

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfv = TfidfVectorizer()
text_data_train = tfv.fit_transform(text_data_train)
text_data_valid = tfv.fit_transform(text_data_valid)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_11.png)

The TfidfVectorizer performs better than the counts most of the time and I have seen that the following parameters for TfidfVectorizer work almost all the time.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfv = TfidfVectorizer(min_df=3,  max_features=None,
      strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
      ngram_range=(1, 2), use_idf=1,smooth_idf=1,sublinear_tf=1,
      stop_words = 'english')
```

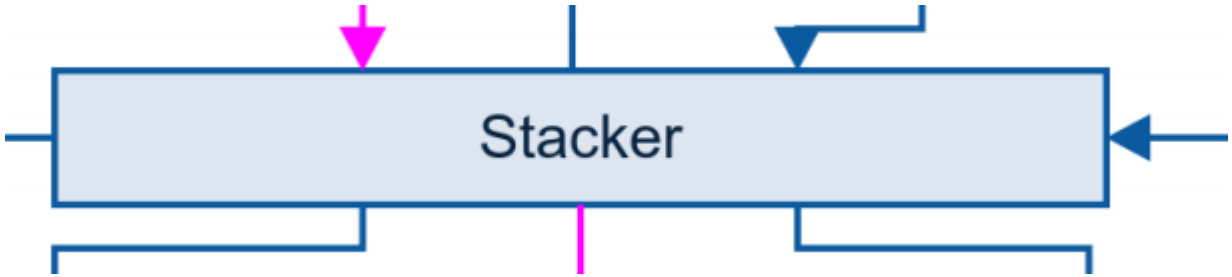
(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_12.png)

If you are applying these vectorizers only on the training set, make sure to dump it to hard drive so that you can use it later on the validation set.

```
import cPickle
cPickle.dump(vectorizer, open('vectorizer.pkl', 'wb'), -1)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_13.png)

Next, we come to the stacker module. Stacker module is not a model stacker but a feature stacker. The different features after the processing steps described above can be combined using the stacker module.



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_14.png)

You can horizontally stack all the features before putting them through further processing by using numpy hstack or sparse hstack depending on whether you have dense or sparse features.

```
import numpy as np
from scipy import sparse

# in case of dense data
X = np.hstack((x1, x2, ...))

# in case data is sparse
X = sparse.hstack((x1, x2, ...))
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_15.png)

And can also be achieved by FeatureUnion module in case there are other processing steps such as pca or feature selection (we will visit decomposition and feature selection later in this post).

```
from sklearn.pipeline import FeatureUnion
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest

pca = PCA(n_components=10)
skb = SelectKBest(k=1)
combined_features = FeatureUnion([("pca", pca), ("skb", skb)])
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_16.png)

Once, we have stacked the features together, we can start applying machine learning models. At this stage only models you should go for should be ensemble tree based models. These models include:

- RandomForestClassifier
- RandomForestRegressor

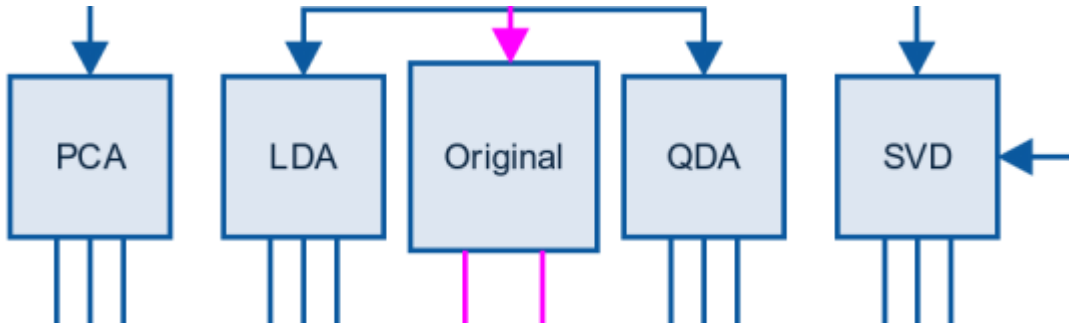
- ExtraTreesClassifier
- ExtraTreesRegressor
- XGBClassifier
- XGBRegressor

We cannot apply linear models to the above features since they are not normalized. To use linear models, one can use Normalizer or StandardScaler from scikit-learn.

These normalization methods work only on dense features and don't give very good results if applied on sparse features. Yes, one can apply StandardScaler on sparse matrices without using the mean (parameter: with_mean=False).

If the above steps give a "good" model, we can go for optimization of hyperparameters and in case it doesn't we can go for the following steps and improve our model.

The next steps include decomposition methods:



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_17.png)

For the sake of simplicity, we will leave out LDA and QDA transformations. For high dimensional data, generally PCA is used decompose the data. For images start with 10-15 components and increase this number as long as the quality of result improves substantially. For other type of data, we select 50-60 components initially (we tend to avoid PCA as long as we can deal with the numerical data as it is).

```
from sklearn.decomposition import PCA

pca = PCA(n_components=12)
pca.fit(xtrain)
xtrain = pca.transform(xtrain)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_18.png)

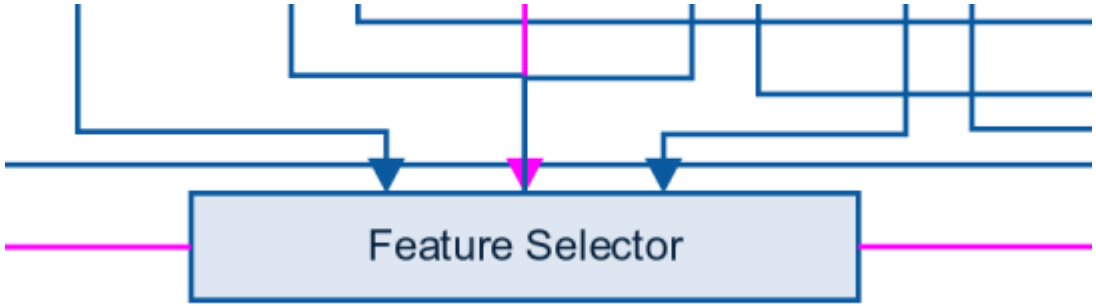
For text data, after conversion of text to sparse matrix, go for Singular Value Decomposition (SVD). A variation of SVD called TruncatedSVD can be found in scikit-learn.

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=120)
svd.fit(xtrain)
xtrain = svd.transform(xtrain)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_decomp.png)

The number of SVD components that generally work for TF-IDF or counts are between 120-200. Any number above this might improve the performance but not substantially and comes at the cost of computing power.

After evaluating further performance of the models, we move to scaling of the datasets, so that we can evaluate linear models too. The normalized or scaled features can then be sent to the machine learning models or feature selection modules.



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_19.png)

There are multiple ways in which feature selection can be achieved. One of the most common way is greedy feature selection (forward or backward). In greedy feature selection we choose one feature, train a model and evaluate the performance of the model on a fixed evaluation metric. We keep adding and removing features one-by-one and record performance of the model at every step. We then select the features which have the best evaluation score. One implementation of greedy feature selection with AUC as evaluation metric can be found here: https://github.com/abhishekkrthakur/greedyFeatureSelection (https://github.com/abhishekkrthakur/greedyFeatureSelection). It must be noted that this implementation is not perfect and must be changed/modified according to the requirements.

Other faster methods of feature selection include selecting best features from a model. We can either look at coefficients of a logit model or we can train a random forest to select best features and then use them later with other machine learning models.

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
clf.fit(X, y)
X_selected = clf.transform(X)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_20.png)

Remember to keep low number of estimators and minimal optimization of hyper parameters so that you don't overfit.

The feature selection can also be achieved using Gradient Boosting Machines. It is good if we use xgboost instead of the implementation of GBM in scikit-learn since xgboost is much faster and more scalable.

```
import xgboost as xgb

params = {}

model = xgb.train(params, dtrain, num_boost_round=100)
sorted(model.get_fscore().items(), key=lambda t: -t[1])
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_21.png)

We can also do feature selection of sparse datasets using RandomForestClassifier / RandomForestRegressor and xgboost.

Another popular method for feature selection from positive sparse datasets is chi-2 based feature selection and we also have that implemented in scikit-learn.

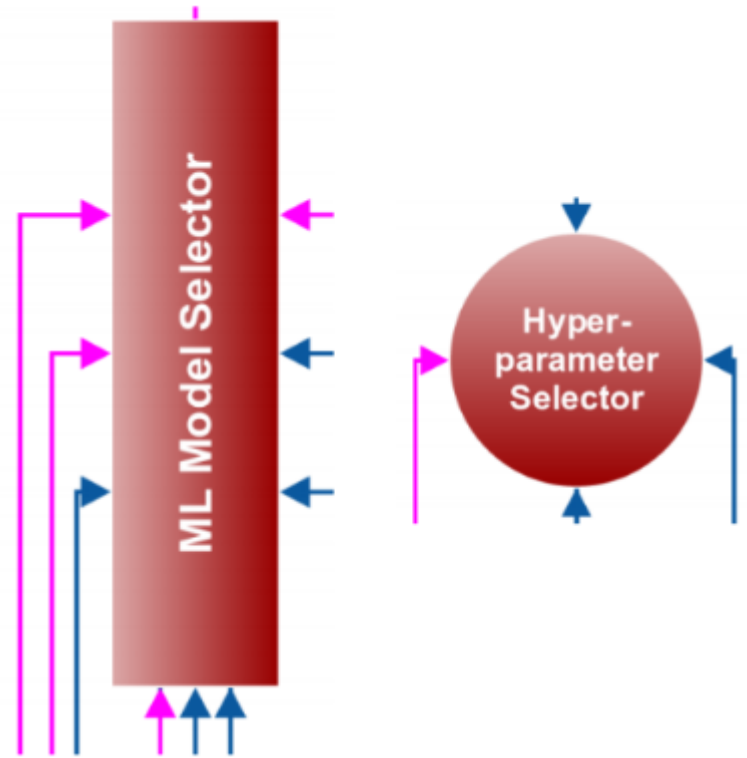
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

skb = SelectKBest(chi2, k=20)
skb.fit_transform(X, y)
```

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_22.png)

Here, we use chi2 in conjunction with SelectKBest to select 20 features from the data. This also becomes a hyperparameter we want to optimize to improve the result of our machine learning models.

Don't forget to dump any kinds of transformers you use at all the steps. You will need them to evaluate performance on the validation set.

Next (or intermediate) major step is model selection + hyperparameter optimization.



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_23.png)

We generally use the following algorithms in the process of selecting a machine learning model:

- **Classification**:
  - Random Forest
  - GBM

- Logistic Regression
- Naive Bayes
- Support Vector Machines
- k-Nearest Neighbors

- **Regression**
  - Random Forest
  - GBM
  - Linear Regression
  - Ridge
  - Lasso
  - SVR

Which parameters should I optimize? How do I choose parameters closest to the best ones? These are a couple of questions people come up with most of the time. One cannot get answers to these questions without experience with different models + parameters on a large number of datasets. Also people who have experience are not willing to share their secrets. Luckily, I have quite a bit of experience too and I'm willing to give away some of the stuff.

Let's break down the hyperparameters, model wise:

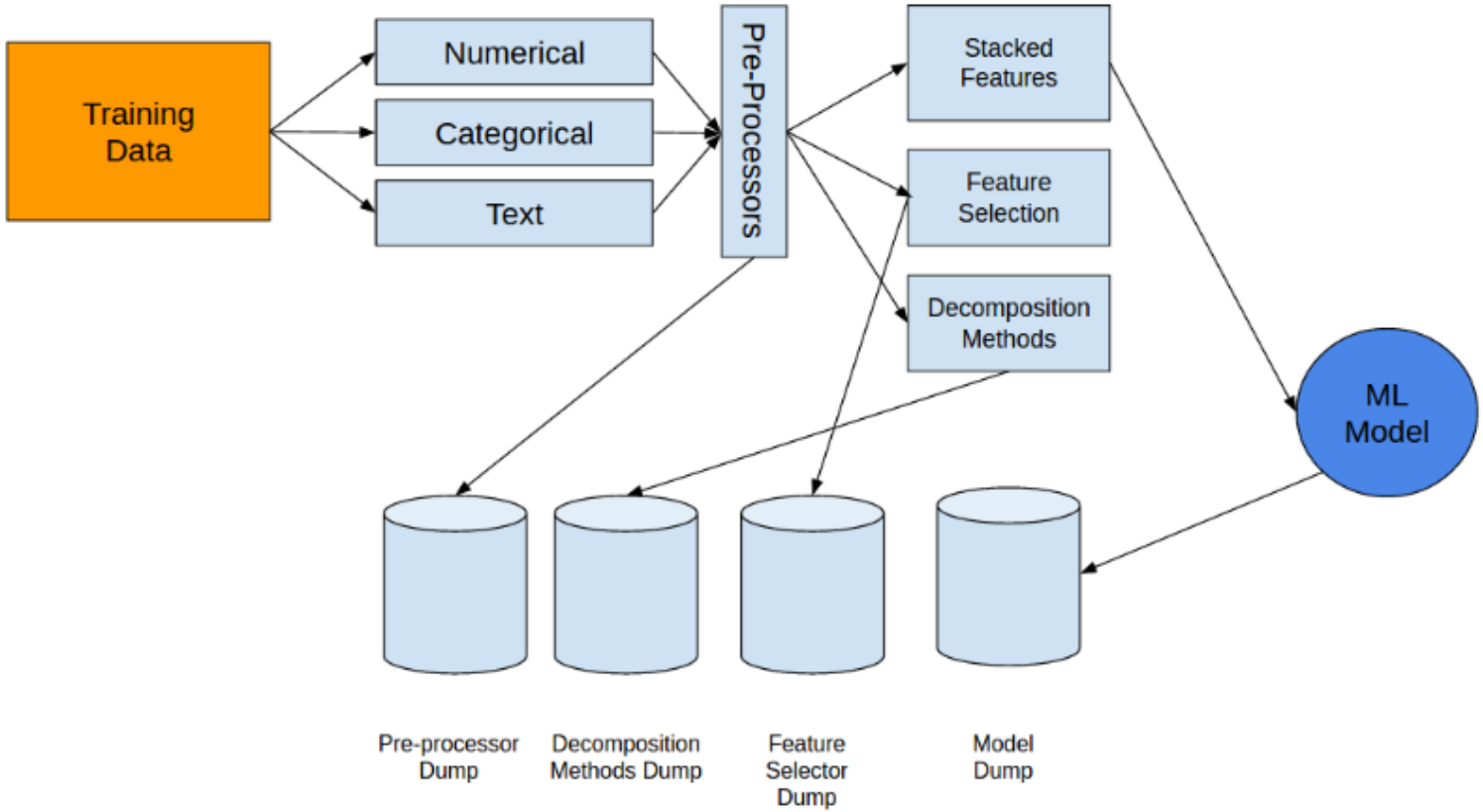
Model	Parameters to optimize	Good range of values
Linear Regression	• fit_intercept • normalize	• True / False • True / False
Ridge	• alpha • Fit_intercept • Normalize	• 0.01, 0.1, 1.0, 10, 100 • True/False • True/False
k-neighbors	• N_neighbors • p	• 2, 4, 8, 16 …. • 2, 3
SVM	• C • Gamma • class_weight	• 0.001, 0.01…..10...100...1000 • 'Auto', RS* • 'Balanced' , None
Logistic Regression	• Penalty • C	• L1 or l2 • 0.001, 0.01…..10...100
Naive Bayes (all variations)	NONE	NONE
Lasso	• Alpha • Normalize	• 0.1, 1.0, 10 • True/False
Random Forest	• N_estimators • Max_depth • Min_samples_split • Min_samples_leaf • Max features	• 120, 300, 500, 800, 1200 • 5, 8, 15, 25, 30, None • 1, 2, 5, 10, 15, 100 • 1, 2, 5, 10 • Log2, sqrt, None
Xgboost	• Eta • Gamma • Max_depth • Min_child_weight • Subsample • Colsample_bytree • Lambda • alpha	• 0.01,0.015, 0.025, 0.05, 0.1 • 0.05-0.1,0.3,0.5,0.7,0.9,1.0 • 3, 5, 7, 9, 12, 15, 17, 25 • 1, 3, 5, 7 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.6, 0.7, 0.8, 0.9, 1.0 • 0.01-0.1, 1.0 , RS* • 0, 0.1, 0.5, 1.0 RS*

(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_24.png)

RS* = Cannot say about proper values, go for Random Search in these hyperparameters.

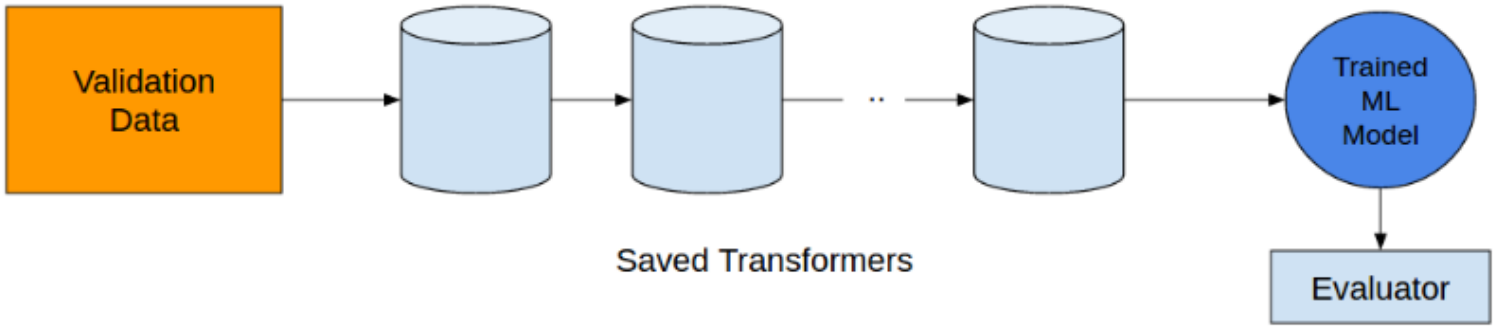
In my opinion, and strictly my opinion, the above models will out-perform any others and we don't need to evaluate any other models.

Once again, remember to save the transformers:



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_25.png)

And apply them on validation set separately:



(http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2016/07/abhishek_26.png)

The above rules and the framework has performed very well in most of the datasets I have dealt with. Of course, it has also failed for very complicated tasks. Nothing is perfect and we keep on improving on what we learn. Just like in machine learning.

Get in touch with me with any doubts: abhishek4 [at] gmail [dot] com

Bio

(http://5047-presscdn.pagely.netdna-

Abhishek Thakur (https://www.kaggle.com/abhishek) works as a Senior Data Scientist on the Data Science team at Searchmetrics Inc (http://www.searchmetrics.com/). At Searchmetrics, Abhishek works on some of the most interesting data driven studies, applied machine learning algorithms and deriving insights from huge amount of data which require a lot of data munging, cleaning, feature engineering and building and optimization of machine learning models.

In his free time, he likes to take part in machine learning competitions and has taken part in over 100 competitions. His research interests include automatic machine learning, deep learning, hyperparameter optimization, computer vision, image analysis and retrieval and pattern recognition.

cdn.com/wp-content/uploads/2016/07/abhishek.png)

ABHISHEK THAKUR (HTTPS://WWW.KAGGLE.COM/ABHISHEK), COMPETITIONS GRANDMASTER.

PYTHON (HTTP://BLOG.KAGGLE.COM/TAG/PYTHON/)

TUTORIAL (HTTP://BLOG.KAGGLE.COM/TAG/TUTORIAL/)

29 Comments **No Free Hunch** **1 Login**

Recommend 51 **Share** **Sort by Best**

Join the discussion…

Bojan Tunguz • 4 months ago

Great post! This is one of the most useful writeups about the general approach to ML that I've come across.

4 • Reply • Share ›

Bruce Robbins • 4 months ago

I agree with the other commentators on the value of this post, if for nothing else, for confirming the “The dirty little secret of big data,” being the fact , “that most data analysts spend the vast majority of their time cleaning and integrating data—not actually analysing it.” At a practical level the overview on hyperparameters optimisation and evaluation is very useful.

3 • Reply • Share ›

Ayush Singh • 4 months ago

Thanks for sharing this informative post!

I have a question abt how to go about choosing the statistical test i.e Chi2 ,ANOVA etc for SelectKbest in sklearn.I have read that Chi is used to see correlation btw Categorical vs Categorical variables and ANOVA for Categorical vs Continuous variables.I see here that you have used Chi2 test for k=20 but there are some features that are continuous like Age,Fare with target variable as Categorical so I am a bit confused

bit confused.

2 ^ v • Reply • Share ›

jdunn ➚ Ayush Singh • a month ago

Must learn anova

^ v • Reply • Share ›

abhi • 4 months ago

Thanks all!

2 ^ v • Reply • Share ›

PANKAJ • 4 months ago

Fantastic post..from the vast jungle of possibilities this is one way to get to results!

2 ^ v • Reply • Share ›

Dhiraj Tandon • 4 months ago

Awsome post Abhishek

2 ^ v • Reply • Share ›

kevinzakka • 4 months ago

Great writeup, and couldn't have come at a better time. Thanks for sharing!

2 ^ v • Reply • Share ›

Steve Joseph • 4 months ago

Thanks. I'm curious where neural networks/deep learning fits into this model?

1 ^ v • Reply • Share ›

abhi ➚ Steve Joseph • 4 months ago

I havent discussed neural nets in this post. But will soon write about parameter tuning and network architecture selection for neural nets.

^ v • Reply • Share ›

Sheshachalam Ratnala ➚ abhi • 4 months ago

I am also a bit curious how it applies to Unsupervised Problem. The post seems to be looking at supervised problem only

1 ^ v • Reply • Share ›

sampath kumar • 4 months ago

Great Post & Thank you.

1 • Reply • Share ›

Trần Đức Nhuận • 4 months ago

Amazing great article guidelines. Thanks Abhishek

1 • Reply • Share ›

Huey Kwik • 24 days ago

Instead of having a separate validation set, why not just use cross-validation?

• Reply • Share ›

stuti awasthi • 2 months ago

Exceptionally written post. Thanks for sharing hyper params optimization generalized rules. Definetly a good read. Thanks

• Reply • Share ›

Adrian Gasinski • 4 months ago

Wow, It is a great article showing professional an practical approach to data mining problems. Thank you very much!

• Reply • Share ›

Sampath sree • 4 months ago

Great post. Thanks for sharing.

• Reply • Share ›

Mohamed Rachidi • 4 months ago

Very good post, Thank you

• Reply • Share ›

alchemication • 4 months ago

Amazing write up, very helpful, thanks!

• Reply • Share ›

Jeff • 4 months ago

I don't think OneHotEncoder is actually necessary to simply using the LabelEncoder. RandomForest can handle integer labels. Also, it's interesting that you used next(iter(kf)) instead of using sklearn.cross_validation.train_test_split (which is a wrapper for next(iter(kf))).

Reply • Share ›

abhi ➚ Jeff • 4 months ago

yes. RandomForest can handle integer labels, but the post isn't just about random forest or GBMSs ;)
i used next(iter(kf)) only to make my simple post a bit complicated.... lol

1 Reply • Share ›

Jeff ➚ abhi • 4 months ago

Fair enough! I'd also be very interested in seeing how you approach data visualization, but that might be another post for another time. Thanks!

Reply • Share ›

Jarad Collier • 4 months ago

So many great tips in here I never knew about or considered! Sincerely, thanks for sharing these ideas.

Reply • Share ›

Yurii Shevchuk • 4 months ago

Thank you for the interesting post!
I want to clarify one thing. You've mentioned that

> We cannot apply linear models to the above features since they are not normalized.

Is it necessary to apply StandardScaler to the data when you try to fit a linear model? Least Squares should work fine without feature scaling assuming that scale of variables is not to big. Otherwise it can cause computational problems.

Reply • Share ›

DS Yu • 4 months ago

This blog is gonna be my small awsome Bible!!

Reply • Share ›

Krish Mahajan • 4 months ago

Thanks a lot

• Reply • Share ›

Nishant Jain • 4 months ago

Thanks Abhishek.. Great Post!!!

• Reply • Share ›

Daniel R Gonzalez • 4 months ago

Great Post!

• Reply • Share ›

StartupFlux • 4 months ago

Great post! Easy to understand and very informative. Thanks for Sharing!

• Reply • Share ›

(https://www.facebook.com/kaggle)

(https://twitter.com/kaggle)