세상에 있는 (거의) 모든 머신러닝 문제 공략법

Tuesday, August 9, 2016

좋은 블로그 게시물이 나와서 번역합니다. 원문은 이곳 을 참고하시기 바랍니다.
Karpathy의 글도 그랬지만 그림은 그대로 퍼오고, 글도 가급적 그대로 번역합니다. 교육 목적으로 활용해주시길 바랍니다.
제가 보충 설명을 하는 경우엔 괄호안에 (**요렇게) 적어놓았습니다.

세상에 있는 (거의) 모든 머신러닝 문제 공략법 (Approaching (Almost) Any Machine Learning Problem)

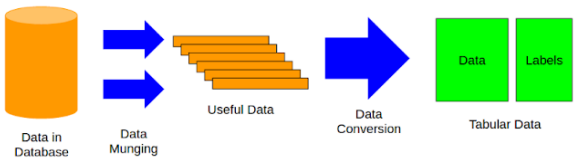
저자: Abshshek Thakur
번역: 최근우

데이터 사이언티스트들은 하루종일 데이터를 다룹니다. 사람에 따라 다르겠지만 60-70%의 시간을 데이터를 전처리하고 원하는 형태로 변환하는데 사용한다는 사람도 있습니다. 그런 과정이 끝나고 나면 실제로 데이터에 머신러닝 알고리즘을 적용합니다. 이 글에서는 간단한 전처리 과정 및 머신러닝을 적용하는 과정을 자세히 다룹니다. 여기에서 소개하는 파이프라인은 제가 백개가 넘는 머신러닝 대회에 참가한 경험을 바탕으로 배운 내용입니다.

파이썬을 사용해 설명드리겠습니다.

데이터

머신러닝을 활용하려면 데이터를 테이블의 형태로 변환해야합니다. 이 과정은 시간도 오래걸리고 힘든 과정입니다.



이 테이블의 각 행(row)은 각각의 데이터 샘플에 해당합니다. 보통 입력 데이터를 X, 레이블 (혹은 출력)을 y로 표현합니다. 레이블은 하나의 열(single column)일 수도 있고 여러 열로 이루어져 있을 수도 있습니다.

레이블의 종류

- 단일 열, 이진수 (분류 문제. 레이블엔 두 개의 카테고리가 존재하고 모든 데이터 샘플은 반드시 둘 중 하나의 카테고리에만 해당하는 경우. 0과 1로 각각 다른 카테고리를 표현할 수 있으므로 이진값을 사용한다.)(**예: 텍스트를 보고 저자의 성별을 맞추는 경우)
- 단일 열, 실수 (회귀 문제. 단일 열이므로 단 하나의 값만 예측하면 된다.)(**예: 텍스트를 보고 저자의 나이를 예측하는 경우)
- 여러 열, 이진수 (분류 문제. 여러 카테고리가 존재하고 각 데이터는 하나의 카테고리에 해당한다.)(**예: 댓글을 보고 작성자가 응원하는 스포츠 팀을 맞추기, 이 경우에 열의 개수는 스포츠 팀의 개수와 같다.)
- 여러 열, 실수 (회귀 문제이나 여러 값을 예측함.)(**예: 의료 검진 데이터를 보고 피검사자가 여러 종류의 암에 걸릴 확률을 예측. 이 경우 열의 개수는 암 종류의 개수와 같다.)
- 다중 레이블 (분류 문제, 각 데이터 샘플이 여러개의 카테고리에 속할 수 있다.)(**예: 음악 신호를 보고 드럼, 기타, 보컬이 있는지 여부를 판단하는 경우. 열의 개수는 3개가 된다.)

평가 방법

어떤 경우든지간에 머신러닝 알고리즘의 평가는 필수입니다. 각 카테고리의 데이터 개수가 크게 차이나는 경우에 ROC AUC를 사용할 수도 있고, 다중 레이블 분류 문제는 카테고리 크로스 엔트로피나 다중레이블 로그 로스 등을 활용합니다.
평가 방법은 상황에 따라 다르기 때문에 여기에선 자세히 다루지 않습니다.

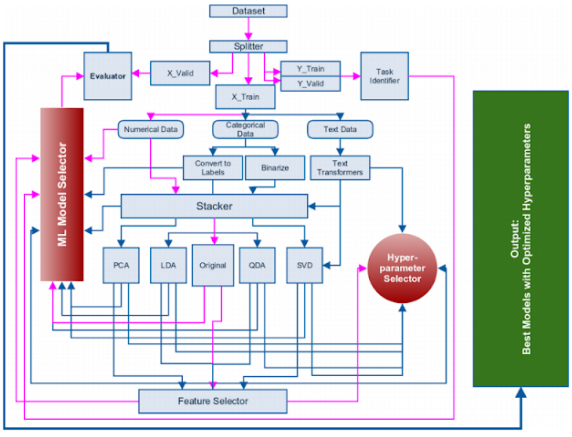
라이브러리

필수적인 라이브러리를 몇 가지 소개합니다.

- 데이터를 보고 간단한 연산을 수행: pandas
- 각종 머신러닝 모델: scikit-learn
- 최강의 그라디언트 부스팅 라이브러리: xgboost
- 뉴럴 네트워크: keras
- 데이터 시각화: matplotlib
- 실시간 모니터링: tqdm

아나콘다 를 사용해 다양한 라이브러리를 쉽게 설치할 수 있습니다. 저는 사용하고있지 않지만 각자 판단하시길 바랍니다.

머신러닝 프레임워크

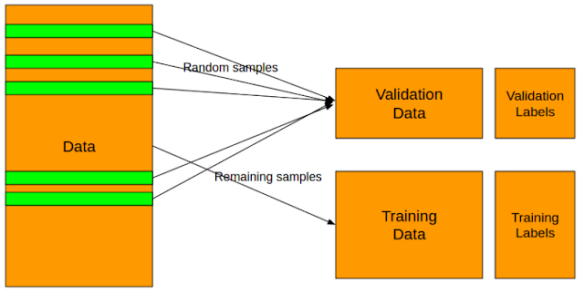


2015년부터 제가 개발중인 머신러닝 자동화 프레임워크의 그림입니다. 이 글에서도 그림과 같은 구조로 작업을 수행합니다.

가장 일반적으로 적용되는 단계를 진한 분홍색 화살표로 그렸습니다. 우선 앞에서 이야기한 전처리 과정을 거쳐 표 형태로 데이터를 정리하고나면 이 단계를 시작할 수 있습니다.

전처리

우선 우리에게 주어진 문제가 어떤 문제인지 파악해야합니다. 이진 분류(0 vs 1)인지, 여러 카테고리중 하나를 고르는 다범주 분류(multi-class classification)인지, 다중 레이블 분류문제인지 아니면 회귀 문제인지를 명확하게 결정해야합니다. 그리고 나면 우리가 가지고 있는 데이터를 학습 셋(training set)과 검증 셋(validation set)으로 나눕니다. 아래 그림을 참고하시길 바랍니다.



(**중요) 두 셋으로 나눌 때에는 반드시 레이블 정보를 사용해야합니다. 분류 문제의 경우 stratified splitting을 사용하시기 바랍니다. 파이썬에선 scikit-learn을 사용하면 됩니다. (**이렇게 하면 두 셋에서 레이블의 분포가 동일하게 유지됩니다.)

```
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]
```

회귀 문제의 경우 그냥 간단하게 K-fold로 셋을 나눠주면 됩니다. 회귀에서도 분포를 유지하는 방법이 있긴 한데 독자여러분이 생각해보시길 바랍니다. (**아니 저기..)

```
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]
```

이 코드에서는 검증 셋의 크기를 전체의 10%로 설정했습니다. 이 값은 가지고 있는 데이터의 크기에 따라 적절히 설정하면 됩니다.

이렇게 데이터를 나눈 뒤에는 두 셋을 동일하게 처리해야합니다. 그리고 절대 학습 셋과 검증 셋에 데이터가 겹치면 안됩니다. 그렇게 할 경우 마치 학습이 아주 잘 된 것처럼 착각할 수 있습니다. 실제로는 엄청난 과적합(overfitting)이 일어나겠죠.

다음 단계는 데이터에 어떤 형태의 값이 있는지 알아내는 것입니다. 대략 숫자, 범주 정보, 그리고 문자열 데이터정도로 나눠서 생각할 수 있습니다. 카글의 타이타닉 데이터셋을 살펴볼까요?

```
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)
```

여기에서는 survival이 레이블이 됩니다. 그리고 각 승객의 좌석 등급, 성별, 탑승한 항구는 범주화해서 생각할 수 있습니다. 나이, 형제자매의 수, 부모/자녀 등은 숫자 정보가 되구요. 마지막으로 이름은 문자열입니다. 이 작업 (**생존 여부를 예측하는 작업)에서는 크게 중요하지 않을 것 같군요.

우선 숫자를 봅시다. 이 값은 특별한 전처리 필요없이 바로 사용할 수 있습니다. 따라서 정규화(normalization) 및 머신러닝 기법을 바로 적용하면 됩니다.

범주 데이터는 두가지 방법으로 처리할 수 있습니다.

- 카테고리를 라벨로 변환

```
from sklearn.preprocessing import LabelEncoder

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

- 카테고리를 이진 변수 (one-hot-vector)(**하나의 성분만 1이고 나머지는 0인 벡터)로 변환

```
from sklearn.preprocessing import OneHotEncoder

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

여기서 OneHotEncoder를 쓰기 전에 우선 LabelEncoder를 적용해야 하는 점을 주의하시기 바랍니다.

타이타닉 데이터셋의 문자열은 문자열 변수의 예로는 별로 좋지 않습니다. 일반적으로 문자열을 어떻게 다루는지 한번 알아보겠습니다.

우선 (**pandas를 쓸 경우) 아래 코드를 써서 데이터에 있는 모든 텍스트를 하나로 이어 붙일 수 있습니다.

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

그리고 여기에 CountVectorizer 또는 TfidfVectorizer를 사용합니다.

```
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)
```

```
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)
```

보통 TfidfVectorizer가 더 잘 작동합니다. (**당연히..) 대체로 아래처럼 셋팅하면 무난하게 사용 가능합니다.

```
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')
```

만일 이 Vectorizer를 트레이닝 데이터에만 적용했다면, 나중에 사용할 수 있도록 하드디스크에 저장을 해놓아야합니다.

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

이제 Stacker 모듈을 봅시다. 여기에서는 다양한 특징값을 합칩니다.
일반적인 데이터는 np.hstack으로 벡터를 이어주면 됩니다.
만일 데이터가 sparse한 경우 (대부분의 값이 0인 경우) scipy.sparse를 사용합니다.

```
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, ...))
```

만일 PCA같은 과정이 포함되어있다면 sklearn.pipeline.FeatureUnion을 쓰면 됩니다.

```
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)])
```

자, 드디어 데이터 전처리가 끝났습니다.
이제 머신러닝 알고리즘을 적용하면 됩니다.

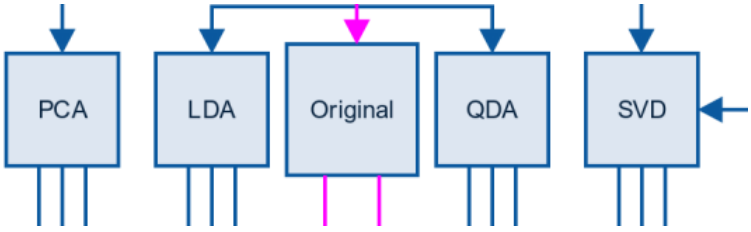
머신러닝 적용
어떤 모델을 적용할까요? 위의 단계를 밟은 경우엔 트리 기반의 알고리즘만 가능합니다. 트리 기반의 모델은 여러가지 종류가 있습니다.

- RandomForestClassifier
- RandomForestRegressor
- ExtraTreesClassifier
- ExtraTreesRegressor
- XGBClassifier
- XGBRegressor

선형 모델을 사용하려면 위의 특징값을 Normalize하는 과정을 거쳐야합니다. sklearn의 Normalizer나 StandardScaler를 사용하면 됩니다. 단, 데이터가 dense한 경우에만 사용이 가능합니다. Sparse한 경우엔 결과가 좋지 않습니다.

위의 트리 모델을 사용해서 그럭저럭 괜찮은 결과가 나왔다면 이제 모델을 자세하게 튜닝할 차례입니다.

다음 단계는 decomposition 입니다.



여기에선 LDA와 QDA는 생략합니다. 고차원 데이터의 경우 PCA(주성분분석)를 쓰면 차원을 효과적으로 줄일 수 있습니다. 이미지의 경우 10-15개의 성분을 사용해보고 차원을 늘려가며 성능을 비교해보십시오. 그 외엔 50-60개정도의 성분으로 시작해보십시오. 단, 값을 그대로 사용해도 괜찮은 상황이라면 굳이 차원을 줄일 필요는 없습니다.

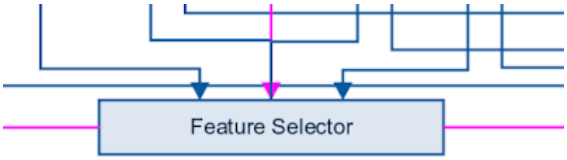
텍스트의 경우는 sparse 행렬에 SVD를 적용합니다.

```
from sklearn.decomposition import PCA

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

```
from sklearn.decomposition import TruncatedSVD

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

보통 120-200정도의 값을 사용합니다. 이보다 큰 값을 쓴다면 연산량에 유의하십시오.

이제 선형 모델도 사용할 수 있도록 특징값을 normalize하거나 re-scaling합니다. (**특징값의 범위를 조절) 그리고 나면 유용한 특징값을 고르는 Feature Selector 단계로 들어갑니다.



특징값을 선정하는 방법은 여러가지가 있습니다. Greedy 알고리즘을 쓸 경우엔, 우선 하나의 특징값을 골라서 모델을 돌리고 여기에 다른 특징값을 더하거나 빼면서 결과를 비교합니다. AUC로 모델을 평가하면서 특징값을 고르는 코드 를 참고하십시오.

아래의 코드는 RandomForest를 이용한 예제입니다. 이렇게 학습한 모델을 저장해놓고 나중에 사용할 수도 있습니다.

```
from sklearn.ensemble import RandomForestClassifier

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

n_opimators 등의 하이퍼파라미터를 너무 크게 설정하면 과적합(overfitting)이 일어나니 주의하십시오.

혹은 xgboost를 사용할 수도 있습니다.

```
import xgboost as xgb

params = {}

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

데이터가 sparse한 경우에도 RandomForestClassifier, RandomForestRegressor, xgboost 등을 사용 가능합니다.

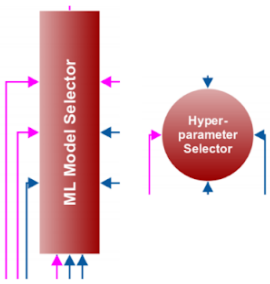
그 외에도 chi-2로 특징값 선택이 가능합니다.

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

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

여기에선 k를 20으로 설정했는데, 이 값 역시 하이퍼파라미터이므로 최적화가 필요합니다.

지금까지 여러 모델을 소개했습니다. 데이터를 처리할 때 이렇게 학습한 모델을 꼭 저장해놓으십시오.

모델 선택 및 최적화
이제 모델을 선택하고 하이퍼파라미터를 최적화하는 일이 남았습니다.



널리 쓰이는 모델은 아래와 같습니다.

- 분류
  - Random Forest
  - GBM
  - Logistic Regression
  - Naive Bayes
  - Support Vector Machines
  - k-Nearest neighbors

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

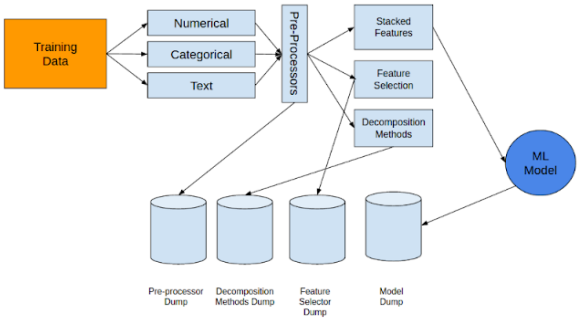
하이퍼 파라미터를 최적화하는 과정은 정해진 공식이 없습니다. 많은 사람들이 이와 관련해 질문을 합니다만 데이터와 모델에 따라 달라지기때문에 한마디로 말하기가 어렵습니다. 또, 몇 가지 팁이 있지만 경험이 많은 사람들은 이 비법을 숨기려고 하기도 합니다. 하지만 제가 알려드리겠습니다.

각 모델별로 어떤 하이퍼파라미터가 있는지 정리하면..

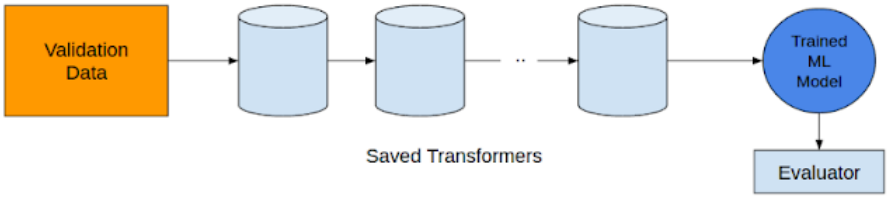
Model	Parameters to optimize	Good range of values
Linear Regression	• fit_intercept • normalize	• True / False • True / False
Ridge	• alpha • Fit_intercept • Normalize	• 0.01, 0.1, 1.0, 10, 100 • True/False • True/False
k-neighbors	• N_neighbors • p	• 2, 4, 8, 16 …. • 2, 3
SVM	• C • Gamma • class_weight	• 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*

위의 값을 임의로 조합한 랜덤 서치를 추천합니다.
제 생각일뿐이지만, 위의 값을 기반으로 찾아본다면 충분할겁니다.

정리
마지막으로 학습 과정을 정리해보면..



그리고 학습 과정에서 저장한 각종 변환(transform)을 다시 불러와서 validation/test set에 사용해야겠죠?



마무리
위의 과정을 잘 따라하면 대체로 아주 좋은 성능을 얻을 수 있을 것입니다. 물론, 잘 되지 않는 경우도 있겠죠. 그러면서 배우는 것이니 열심히 해보시길 바랍니다.