

Random Forest in Python

실용적인 End-to-End Machine Learning 예제

머신러닝을 시작하기에 더할 나위 없이 좋은 시기입니다. 온라인에서 사용할 수 있는 학습 리소스, 상상할 수 있는 모든 알고리즘을 구현한 무료 오픈 소스 도구, **AWS**와 같은 클라우드 서비스를 통한 저렴한 컴퓨팅 성능을 갖춘 머신러닝은 진정으로 인터넷에 의해 대중화된 분야입니다. 노트북에 접근할 수 있고 배우려는 의지가 있는 사람은 누구나 몇 분 안에 최첨단 알고리즘을 시험해 볼 수 있습니다. 조금 더 시간을 내면 일상생활이나 직장에서 도움이 되는 실용적인 모델을 개발할 수 있습니다. (또는 머신러닝 분야로 뛰어들어 경제적 이익을 거두십시오) 이 게시물은 강력한 **random forest** 머신러닝 모델의 **end-to-end** 구현을 안내합니다. 이것은 랜덤 포레스트에 대한 나의 개념적 설명을 보완하기 위한 것이지만, 의사 결정 트리와 랜덤 포레스트에 대한 기본 아이디어를 가지고 있는 한 완전히 독자적으로 읽을 수 있습니다. [이 게시물](#)은 여기에 구축된 모델을 개선할 방법을 자세히 설명합니다.

물론 여기에는 **Python** 코드가 있습니다. 누구에게나 친밀하지는 않습니다. 그러나 오늘날 사용 가능한 리소스를 통해 머신러닝이 얼마나 접근 가능한지 보여줍니다! 데이터가 포함된 전체 프로젝트는 [GitHub](#)에서 사용할 수 있으며 [데이터 파일](#)과 **Jupyter** 노트북은 **Google** 드라이브에서도 다운로드 할 수 있습니다. **Python**이 설치된 노트북과 **Jupyter** 노트북을 시작할 수 있는 기능만 있으면 됩니다. (**Python**을 설치하고 **Jupyter** 노트북을 실행하려면 [이 가이드](#)를 확인하세요) 여기에서는 몇 가지 필수 머신러닝 주제가 있고, 관심 있는 사람들을 위해 더 많은 것을 배울 수 있는 리소스를 제공하고 명확하게 하려고 노력할 것입니다.

문제 서론

우리가 해결해야 할 문제는 1년간의 과거 날씨 데이터를 사용하여 도시의 내일 최고 기온을 예측하는 것입니다. 여기선 워싱턴주 시애틀을 사용하고 있지만, [NOAA Climate Data Online tool](#)을 사용하여 자신의 도시에 대한 데이터를 자유롭게 찾을 수 있습니다. 우리는 일기 예보에 접근할 수 없는 것처럼 행동할 것입니다. (게다가 다른 사람에게 의존하는 것보다 우리 자신이 예측하는 것이 더 재미있습니다) 우리가 접근 할 수 있는 것은 1년의 과거 최고 기온, 지난 이틀 동안의 기온, 항상 날씨에 대한 모든 것을 알고 있다고 주장하는 친구의 추정치입니다. 이것은 지도, 회귀 머신러닝 문제입니다. 예측하려는 **features**(도시 데이터)와 **targets**(기온)이 모두 있으므로 지도 학습됩니다. 훈련 중에 랜덤 포레스트에 **features**과 **targets**를 모두 제공하고 데이터를 예측에 매핑하는 방법을 배워야 합니다. 또한 **targets**값이 연속적이기 때문에 회귀 작업입니다. (분류의 분리된 클래스와 반대) 이것이 우리가 필요로 하는 거의 모든 배경이니 시작하겠습니다!

Roadmap

프로그래밍을 시작하기 전에, 간단한 가이드를 배치해야 합니다. 다음 단계들은 모든 머신러닝 **workflow**의 기초가 됩니다.

1. 문제를 정하고 필요한 데이터를 결정합니다.
2. 접근 가능한 형식으로 데이터 수집
3. 필요에 따라 누락 된 데이터 / 이상을 식별하고 수정합니다.
4. 머신러닝 모델을 위한 데이터 준비
5. 목표를 초과하는 **baseline** 모델을 설정하십시오.
6. 훈련 데이터로 모델 훈련
7. 테스트 데이터로 예측
8. 예측을 **test set targets**와 비교하고 성능 메트릭 계산
9. 성능이 만족스럽지 않으면 모델을 조정하거나 더 많은 데이터를 얻거나 다른 모델링 기술을 시도하십시오.
10. 모델을 해석하고 결과를 시각적 및 수치로 보고

Step 1은 이미 진행했습니다! 질문이 있습니다: "우리 도시의 내일 최고 기온을 예측할 수 있습니까?" 그리고 우리는 지난해 워싱턴주 시애틀에서 과거 최고 기온에 접근할 수 있다는 것을 알고 있습니다.

데이터 취득

먼저 데이터가 필요합니다. 실제 예를 사용하기 위해 2016년부터 NOAA 기후 데이터 온라인 도구를 사용하여 워싱턴주 시애틀의 날씨 데이터를 검색했습니다. 일반적으로 데이터 분석에 걸리는 시간의 약 80%는 데이터를 정리하고 검색하는 데 사용되지만 고품질 데이터 소스를 찾으면 이 작업량을 줄일 수 있습니다. NOAA 도구는 놀라울 정도로 사용하기 쉽고 기존 데이터는 Python 또는 R과 같은 언어로 구문 분석할 수 있는 CSV 파일로 다운로드 할 수 있습니다. 따라가고자 하는 사람들을 위해 [전체 데이터 파일](#)을 다운로드 할 수 있습니다.

다음 Python 코드는 CSV 데이터를 로드하고 데이터 구조를 표시합니다.

Pandas는 데이터 조작에 사용됩니다.

```
import pandas as pd
```

데이터를 읽고 처음 5개 행을 표시합니다.

```
features = pd.read_csv('temps.csv')
```

```
features.head(5)
```

	year	month	day	week	temp_2	temp_1	average	actual	friend
0	2016	1	1	Fri	45	45	45.6	45	29
1	2016	1	2	Sat	44	45	45.7	44	61
2	2016	1	3	Sun	45	44	45.8	41	56
3	2016	1	4	Mon	44	41	45.9	40	53
4	2016	1	5	Tues	41	40	46.0	44	41

각 행이 하나의 관측치를 형성하고, 열은 변수값을 갖는 깔끔한 데이터 형식입니다.

다음은 열에 대한 설명입니다:

- year : 모든 데이터는 2016년
- month : 해당 연도의 월
- day
- week : 문자열로 된 요일
- temp_2 : 2일 전 최고기온
- temp_1 : 1일 전 최고 기온
- average : 과거 평균 최고 기온
- actual : 실제 최고 기온
- friend : 친구의 예측($\text{average} \pm 20$ 의 랜덤 값)

이상/누락된 데이터 식별

데이터의 차원을 살펴보면 행이 348개 뿐이며 2016년에 있었던 366일과는 일치하지 않습니다. NOAA의 데이터에 몇 일이 누락되었습니다. 이것은 실제로 수집된 데이터가 완벽하지 않을 수 있다는 것을 상기시켜줍니다. 누락된 데이터는 잘못된 데이터 또는 이상 값과 같이 분석에 영향을 미칠 수 있습니다. 이 예제에서는 데이터 품질이 좋으므로 누락된 데이터는 큰 영향을 미치지 않습니다. 또한 8개의 features와 1개의 target(actual)을 나타내어 총 9개의 열이 있음을 알 수 있습니다.

```
print('The shape of our features is:', features.shape)
The shape of our features is: (348, 9)
```

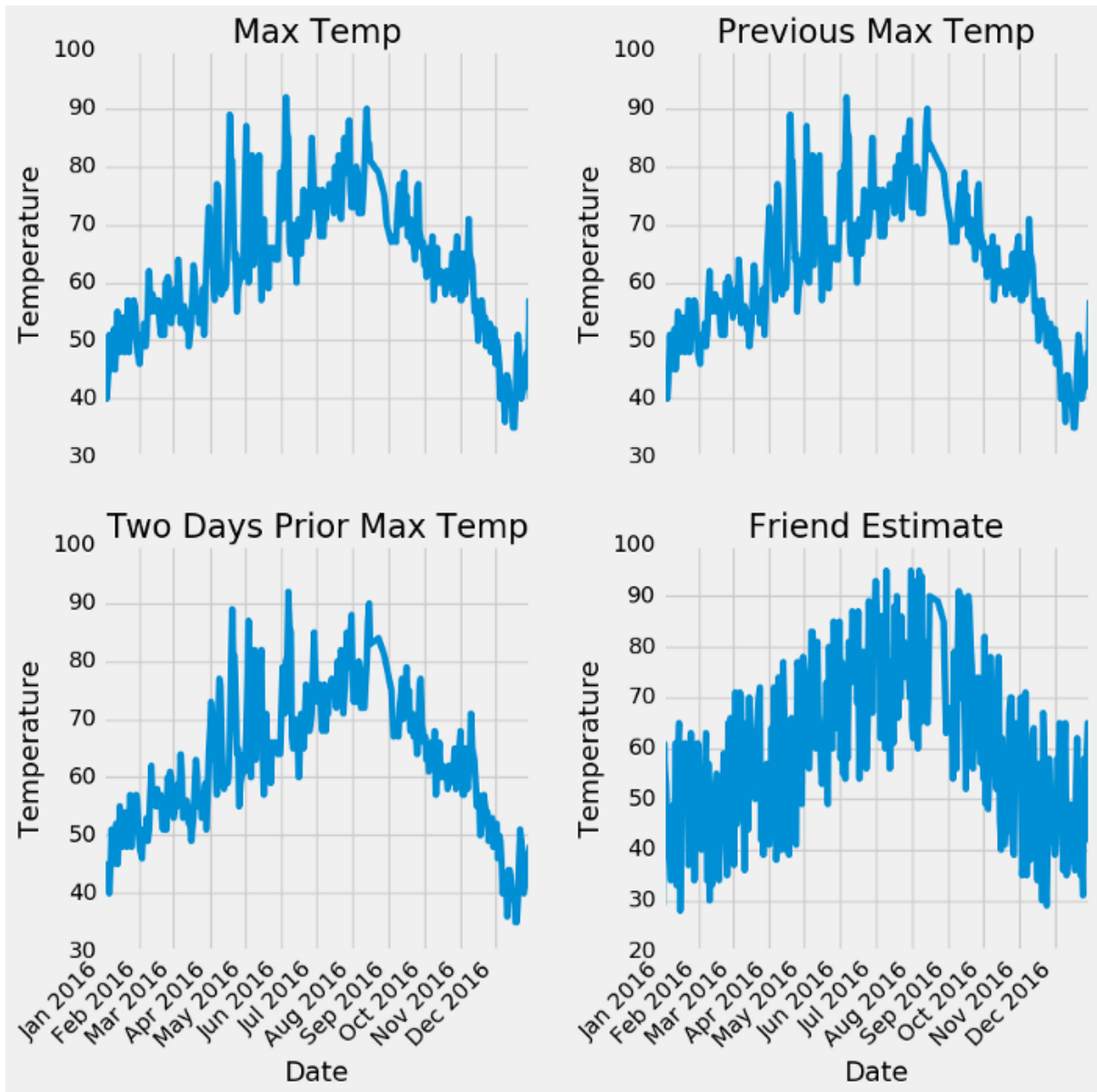
이상을 식별하기 위해 요약 통계를 빠르게 계산할 수 있습니다.

```
# 각 열에 관한 기술 통계
features.describe()
```

	year	month	day	temp_2	temp_1	average	actual	friend
count	348.0	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000
mean	2016.0	6.477011	15.514368	62.511494	62.560345	59.760632	62.543103	60.034483
std	0.0	3.498380	8.772982	11.813019	11.767406	10.527306	11.794146	15.626179
min	2016.0	1.000000	1.000000	35.000000	35.000000	45.100000	35.000000	28.000000
25%	2016.0	3.000000	8.000000	54.000000	54.000000	49.975000	54.000000	47.750000
50%	2016.0	6.000000	15.000000	62.500000	62.500000	58.200000	62.500000	60.000000
75%	2016.0	10.000000	23.000000	71.000000	71.000000	69.025000	71.000000	71.000000
max	2016.0	12.000000	31.000000	92.000000	92.000000	77.400000	92.000000	95.000000

데이터 요약

비정상적으로 나타나는 데이터 포인트가 없고, 측정 열에 0이 없습니다. 데이터의 품질을 확인하는 또 다른 방법은 기본 plots를 만드는 것입니다. 종종 숫자보다 그래프에서 이상을 발견하기가 더 쉽습니다. 여기서 실제 코드를 생략했습니다. 왜냐하면 plotting은 파이썬이 직관적이지 않기 때문입니다. 완전한 구현은 노트북을 참조하십시오. (훌륭한 data scientist처럼 Stack Overflow에서 plotting 코드를 거의 복사하여 붙여넣었습니다.)



정량적 통계와 그래프를 살펴보면, 높은 품질의 데이터라는 확신을 가질 수 있습니다. 명확한 이상치가 없으며 누락 된 점이 몇 개 있더라도 분석에서 크게 벗어나지 않습니다.

데이터 준비


안타깝게도 원시 데이터를 모델에 입력하고 답을 반환할 수 있는 시점은 아닙니다! 데이터를 기계가 이해할 수 있는 용어로 변환하려면 약간의 수정이 필요합니다. 기본적으로 행과 열이 있는 **Excel** 스프레드시트인 데이터 프레임이라는 구조에 의존하여 데이터를 조작하는 **Python** 라이브러리 **Pandas**를 사용합니다.

데이터 준비를 위한 정확한 단계는 사용된 모델과 수집된 데이터에 따라 다르지만 모든 머신러닝 애플리케이션에는 어느 정도의 데이터 조작이 필요합니다.

One-Hot Encoding

첫 번째 단계는 데이터의 **One-Hot Encoding**입니다. 이 프로세스는 요일과 같은 범주형 변수를 사용하여 임의의 순서 없이 숫자 표현으로 변환합니다. 요일은 항상 사용하기 때문에 직관적입니다. '월'이 주중의 첫날을 의미한다는 사실을 모르는 사람은 (아마도) 절대 찾을 수 없습니다. 그러나 기계에도 직관적인 지식이 없습니다. 컴퓨터가 아는 것은 숫자이며 머신러닝을 위해서는 이를 수용해야 합니다. 요일을 숫자 1-7에 간단히 매핑할 수 있지만, 일요일의 숫자 값이 크기 때문에 알고리즘이 일요일에 더 중요하게 될 수 있습니다. 대신 평일의 단일 열을 이진 데이터의 7개 열로 변경합니다. 이것은 그림으로 잘 설명되어 있습니다. **One-Hot Encoding**은 다음을 수행합니다.

week	
Mon	
Tue	
Wed	
Thu	
Fri	



Mon	Tue	Wed	Thu	Fri
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

따라서 데이터 포인트가 수요일이면, 수요일 열에는 1이 있고 다른 모든 열에는 0이 있습니다. 이 과정은 **pandas**에서 한 줄로 할 수 있습니다!

```
# pandas.get_dummies를 사용하여 데이터를 원-핫 인코딩
```

```
features = pd.get_dummies(features)
```

```
# 6열부터 처음 5행을 표시합니다.
```

```
features.iloc[:,5:].head(5)
```

	year	month	day	temp_2	temp_1	average	actual	friend	week_Fri	week_Mon	week_Sat	week_Sun	week_Thurs	week_Tues	week_Wed
0	2016	1	1	45	45	45.6	45	29	1	0	0	0	0	0	0
1	2016	1	2	44	45	45.7	44	61	0	0	1	0	0	0	0
2	2016	1	3	45	44	45.8	41	56	0	0	0	1	0	0	0
3	2016	1	4	44	41	45.9	40	53	0	1	0	0	0	0	0
4	2016	1	5	41	40	46.0	44	41	0	0	0	0	0	1	0

원-핫 인코딩 후 데이터

데이터의 모양은 이제 **349 x 15**이고 모든 열은 숫자입니다. 알고리즘이 좋아하는 방식입니다!

Features and Targets and Convert Data to Arrays

이제 데이터를 **features**와 **targets**로 분리해야 합니다. **label**이라고도 불리는 **target**은 우리가 예측하려는 값입니다. 이 경우 실제 최고 기온과 **features**는 모델이 예측을 수행하는 데 사용하는 모든 열입니다. 또한 **Pandas** 데이터 프레임을 **Numpy**로 변환합니다. 그것이 알고리즘이 작동하는 방식이기 때문입니다. (나중에 시각화할 수 있도록 **features**의 이름인 열 머리글을 목록에 저장합니다.)

```
# numpy를 사용하여 배열로 변환
import numpy as np

# labels는 우리가 예측하려는 값입니다.
labels = np.array(features['actual'])

# features에서 labels 제거
# axis = 1은 열을 나타냅니다.
features = features.drop('actual', axis = 1)

# 나중에 사용하기 위해 features 이름 저장
feature_list = list(features.columns)

# numpy 배열로 변환
features = np.array(features)
```

Training and Testing Sets

데이터 준비의 마지막 단계는 다음과 같습니다: 데이터를 훈련 및 테스트 세트로 나눕니다. 학습 중(Training)에 모델이 답 (이 경우 **actual**)을 '보도록' 허용하므로 **features**에서 기온을 예측하는 방법을 학습할 수 있습니다. 모든 **features**와 **target** 값 사이에 약간의 관계가 있을 것으로 예상하며 모델의 역할은 학습 중에 이 관계를 배우는 것입니다. 그런 다음 모델을 평가할 때가 되면 우리는 (답이 아닌) **features**에만 접근하는 테스트 세트에서 예측합니다! 테스트 세트에 실제 답이 있으므로 이러한 예측을 실제 값과 비교하여 모델이 얼마나 정확한지 판단할 수 있습니다. 일반적으로 모델을 훈련할 때 데이터를 훈련 및 테스트 세트로 무작위로 나누어 모든 데이터 포인트의 표현을 얻습니다. (만약, 첫 9개월을 훈련한 다음 나머지 3개월을 예측에 사용한 경우, 지난 3개월 동안의 데이터를 보지 못했기 때문에 알고리즘이 제대로 작동하지 않습니다) 재현 가능한 결과를 위해 분할을 실행할 때마다 결과가 같다는 것을 의미하는 **random state**를 42로 설정합니다.

다음 코드는 데이터 세트를 다른 한 줄로 나눕니다 :

```
# Skicit-learn을 사용하여 데이터를 훈련 및 테스트 세트로 분할
from sklearn.model_selection import train_test_split

# 데이터를 훈련 및 테스트 세트로 분할
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size
= 0.25, random_state = 42)
```

모든 데이터의 모양을 살펴보고 모든 것을 올바르게 수행했는지 확인할 수 있습니다. **training features** 열 수는 **testing features** 열 수와 일치하고 행 수는 각 **training** 및 **testing features** 및 **labels**와 일치해야 합니다.

```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

Training Features Shape: (261, 14)

Training Labels Shape: (261,)

Testing Features Shape: (87, 14)

Testing Labels Shape: (87,)

모든 것이 제대로 된 것처럼 보입니다! 요약하자면 데이터를 머신러닝에 허용되는 형식으로 가져오기 위해 다음을 수행합니다:

1. 원-핫 인코딩된 범주형 변수
2. 데이터를 **features** 및 **labels**로 분할
3. 배열로 변환
4. 데이터를 **training** 및 **testing sets**로 분할

초기 **data set**에 따라 이상치 제거, 결측값 대체, 시간 변수를 주기적 표현으로 변환하는 것과 같은 추가 작업이 필요할 수 있습니다. 이러한 단계는 처음에는 임의로 보일 수 있지만 일단 기본 **workflow**를 얻으면 일반적으로 모든 머신러닝 문제에서 같습니다. 사람이 읽을 수 있는 데이터를 머신러닝 모델에서 이해할 수 있는 형태로 만드는 것이 전부입니다.

Baseline 설정

예측을 평가하기 전에, 우리는 모델을 능가하기를 희망하는 합리적인 측정 기준을 설정해야 합니다. 모델이 **baseline**을 개선할 수 없다면 실패가 될 것이며 다른 모델을 시도하거나 머신러닝이 문제에 적합하지 않다는 것을 인정해야 합니다. 이 경우에 대한 **baseline** 예측은 과거 최고 기온 평균일 수 있습니다. 다시 말해서, 우리의 **baseline**은 우리가 종일 평균 최고 기온을 단순히 예측했을 때 우리가 얻을 수 있는 오차이다.

```
# baseline 예측은 과거 평균입니다.  
baseline_preds = test_features[:, feature_list.index('average')]  
  
# baseline errors 및 평균 baseline errors 출력  
baseline_errors = abs(baseline_preds - test_labels)  
print('Average baseline error: ', round(np.mean(baseline_errors), 2))
```

Average baseline error: 5.06 degrees.

이제 우리의 목표가 있습니다! 평균 오차 5도를 이길 수 없다면 접근 방식을 재고해야 합니다.

Train Model

모든 데이터 준비 작업을 마친 후 **Scikit-learn**을 사용하여 모델을 만들고 학습시키는 것은 매우 간단합니다. **skicit-learn**에서 랜덤 포레스트 회귀 모델을 가져와서 모델을 인스턴스화 한 다음 훈련 데이터에 모델을 **fit**(**scikit-learn**의 훈련 이름) 합니다. (재현 가능한 결과를 위해 다시 임의 상태 설정)이 전체 프로세스는 **scikit-learn**에서 단 3줄입니다!

```
# 우리가 사용하는 모델 Import
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
# 1000 개의 의사 결정 트리로 모델 인스턴스화
```

```
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
```

```
# 훈련 데이터로 모델 훈련
```

```
rf.fit(train_features, train_labels)
```

Test Set로 예측하기

이제 모델은 **features**와 **targets** 간의 관계를 학습하도록 훈련되었습니다. 다음 단계는 모델이 얼마나 좋은지 알아내는 것입니다! 이를 위해 테스트 **features**로 예측을 합니다. (모델은 테스트 답변을 볼 수 없습니다) 그런 다음 예측을 답변과 비교합니다. 회귀를 수행할 때 일부 답변은 낮고 일부는 높을 것으로 예상하므로 절대 오차를 사용해야 합니다. 우리는 평균 예측이 실제 값에서 얼마나 멀리 떨어져 있는지에 관심이 있으므로 절대값을 취합니다. (**baseline**을 설정할 때도 마찬가지임)

모델을 사용하여 예측하는 것은 **Skicit-learn**의 또 다른 한 줄 명령입니다.

```
# 테스트 데이터로 forest의 예측 방법 사용
```

```
predictions = rf.predict(test_features)
```

```
# 절대 오차 계산
```

```
errors = abs(predictions - test_labels)
```

```
# 평균 절대 오차 출력 (mae)
```

```
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 3.83 degrees.
```

우리의 평균 추정치는 **3.83**도 차이가 있습니다. 이는 **baseline**보다 평균 1도 이상 향상되었습니다. 이것은 중요하지 않은 것 같지만, 분야와 문제에 따라 **baseline**보다 거의 **25%** 더 좋습니다. 이것은 회사에 수백만 달러의 이익을 보게 할 수 있습니다.

성능 Metrics 결정

100%에서 뺀 mean average percentage error를 사용하여 정확도를 계산하여 예측할 수 있습니다.

```
# mean absolute percentage error(MAPE) 계산
```

```
mape = 100 * (errors / test_labels)
```

```
# 계산 및 표시 정확도
```

```
accuracy = 100 - np.mean(mape)
```

```
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 93.99 %.
```

꽤 좋아 보이네요! 우리 모델은 94%의 정확도로 시애틀에서 다음날 최고 기온을 예측하는 방법을 배웠습니다.

필요한 경우 모델 개선

일반적인 머신러닝 workflow는 hyperparameter 튜닝을 하게 될 것입니다.

이것은 "성능 향상을 위한 설정 조정"을 의미하는 복잡한 문구입니다. (이 설정은 훈련 중에 학습한 모델 매개 변수와 구별하기 위해

hyperparameters라고 합니다) 이를 수행하는 가장 일반적인 방법은 설정이

다른 여러 모델을 만들고 같은 유효성 validation set에서 모두 평가한 다음

어느 것이 가장 효과적인지 확인하는 것입니다. 물론 이것은 수작업으로

수행하는 지루한 프로세스이며, Skicit-learn에는 이 프로세스를 수행하는

자동화 된 방법이 있습니다. hyperparameter 튜닝은 이론 기반보다

엔지니어링에 더 가깝습니다. 관심 있는 사람이라면 누구나 [문서](#)를

확인하고 실습해 보길 권장합니다! 이 문제는 94%의 정확도로 만족하지만,

첫 번째로 제작된 모델은 생산을 위한 모델이 될 수 없다는 점을

명심하십시오.

모델 해석 및 결과 보고

이 시점에서 우리는 우리 모델이 훌륭하다는 것을 알고 있지만 거의 [black box](#)입니다. 훈련을 위해 **Numpy** 배열을 제공하고, 예측을 요청하고, 예측을 평가하고, 합리적인지 확인합니다. 질문: 이 모델은 어떻게 값에 도달합니까? 랜덤 포레스트의 내부에 접근하는 두 가지 방법이 있습니다: 첫째, 숲에 있는 하나의 나무를 볼 수 있습니다. 둘째, 설명 변수의 **feature** 중요성을 볼 수 있습니다.

단일 의사 결정 트리 시각화

Skicit-learn에서 **Random Forest** 구현의 가장 멋진 부분 중 하나는 실제로 **forest**의 모든 나무를 검사할 수 있다는 것입니다. 하나의 나무를 선택하고, 전체 나무를 이미지로 저장합니다.

다음 코드는 숲에서 하나의 나무를 가져와 이미지로 저장합니다.

```
# 시각화에 필요한 도구 Import
from sklearn.tree import export_graphviz
import pydot

# 숲에서 나무 한 그루를 꺼내
tree = rf.estimators_[5]

# 이미지를 dot file로 Export
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True,
precision = 1)

# dot file을 사용하여 그래프 생성
(graph, ) = pydot.graph_from_dot_file('tree.dot')

# png 파일에 그래프 쓰기
graph.write_png('tree.png')
```




Forest의 단일 전체 의사 결정 트리

와! 그것은 **15개의 층**을 가진 상당히 광대한 나무처럼 보입니다. (실제로 이것은 제가 봤던 나무에 비해 상당히 작은 나무입니다) 이 이미지를 직접 다운로드하여 더 자세히 살펴볼 수 있지만, 작업을 더 쉽게 하도록 숲의 나무 깊이를 제한하여 이해할 수 있는 이미지를 만들겠습니다.

나무의 깊이를 3단계로 제한

```
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
```

```
rf_small.fit(train_features, train_labels)
```

작은 나무 추출

```
tree_small = rf_small.estimators_[5]
```

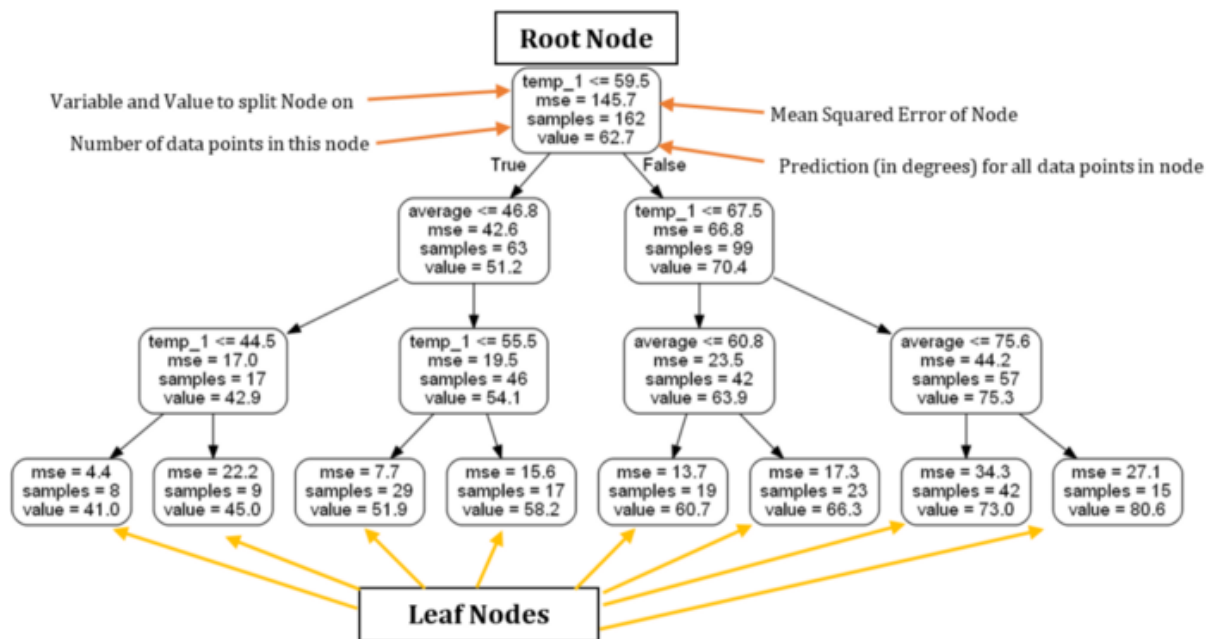
나무를 png 이미지로 저장

```
export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = feature_list, rounded = True, precision = 1)
```

```
(graph, ) = pydot.graph_from_dot_file('small_tree.dot')
```

```
graph.write_png('small_tree.png')
```

다음은 레이블 주석이 달린 축소 된 크기의 트리입니다.



이 트리만을 기반으로 새로운 데이터 포인트에 대해 예측을 할 수 있습니다. 2017년 12월 27일 수요일을 예측하는 예를 들어 보겠습니다. (실제) 변수는 다음과 같습니다: $\text{temp}_2 = 39$ / $\text{temp}_1 = 35$ / $\text{average} = 44$ / $\text{friend} = 30$. 루트 노드에서 시작하고 $\text{temp}_1 \leq 59.5$ 이므로 첫 번째 대답은 True입니다. 왼쪽으로 이동하여 두 번째 질문을 만나면 그것 또한 $\text{average} \leq 46.8$ 로 True입니다. 왼쪽 아래로 이동하고 $\text{temp}_1 \leq 44.5$ 로 True인 세 번째이자 마지막 질문으로 이동합니다. 따라서 leaf node의 값으로 표시된 대로 최고 기온에 대한 추정치는 41.0 도라고 결론을 내립니다. 흥미로운 점은 261개의 훈련 데이터 포인트가 있음에도 불구하고 root node에는 162개의 샘플만 있다는 것입니다. 이는 forest의 각 트리가 데이터 포인트의 무작위 하위 집합으로 훈련을 받았기 때문입니다. (bootstrap aggregating의 줄임말인 bagging이라고 함) (대체로 샘플링을 끄고 bootstrap을 설정하여 모든 데이터 포인트를 사용할 수 있습니다. = forest를 만들 때 False) 데이터 포인트의 Random 샘플링과 트리의 각 노드에서 features 하위 집합의 Random 샘플링이 결합한 모델을 'Random' forest라고 합니다.

또한 트리에는 실제로 예측에 사용한 변수가 2개뿐입니다! 이 특정 의사 결정 트리에 따르면 나머지 features는 예측에 중요하지 않습니다. month, day, 친구의 예측(friend)은 내일 최고 기온을 예측하는 데 전혀 쓸모가 없습니다! simple tree에 따르면 중요한 정보는 1일 전의 기온(temp_1)과

과거 평균(**average**)입니다. 트리를 시각화하는 것은 문제에 대한 도메인 지식을 증가시켰고, 이제 예측하라는 요청을 받는다면 어떤 데이터를 찾아야 할지 알게 되었습니다!

중요 변수

전체 랜덤 포레스트에 있는 모든 변수의 유용성을 정량화하기 위해 변수의 상대적 중요성을 살펴볼 수 있습니다. **Skicit-learn**에서 반환된 중요도는 특정 변수를 포함하면 예측이 얼마나 향상되는지 나타냅니다. 중요도에 대한 실제 계산은 이 게시물의 범위를 벗어나지만, 숫자를 사용하여 변수 간의 상대적 비교를 수행할 수 있습니다.

여기에 있는 코드는 **Python** 언어의 여러 기교, 즉 **list comprehensive**, **zip**, **sorting**, **argument unpacking**을 활용합니다. 지금은 이것을 이해하는 것이 그다지 중요하지 않지만, **Python**에 능숙해지려면 무기고에 있어야 하는 도구입니다!

```
# Get numerical feature importances
```

```
importances = list(rf.feature_importances_)
```

```
# 변수와 중요도가 있는 튜플 목록
```

```
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(feature_list, importances)]
```

```
# 가장 중요한 순서로 feature 중요도 정렬
```

```
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
```

```
# Print out the feature and importances
```

```
[print("Variable: {:20} Importance: {}".format(*pair)) for pair in feature_importances]
```

Variable: temp_1	Importance: 0.7
Variable: average	Importance: 0.19
Variable: day	Importance: 0.03
Variable: temp_2	Importance: 0.02
Variable: friend	Importance: 0.02
Variable: month	Importance: 0.01
Variable: year	Importance: 0.0

Variable: week_Fri	Importance: 0.0
Variable: week_Mon	Importance: 0.0
Variable: week_Sat	Importance: 0.0
Variable: week_Sun	Importance: 0.0
Variable: week_Thurs	Importance: 0.0
Variable: week_Tues	Importance: 0.0
Variable: week_Wed	Importance: 0.0

list의 맨 위에는 전날 최고 기온인 **temp_1**이 있습니다. 이것은 하루의 최고 기온을 예측하는 가장 좋은 변수가 전날의 최고 기온(**temp_1**)이라는 것을 말해줍니다. 두 번째로 중요한 요소는 과거 평균 최고 기온(**average**)이며, 그다지 놀랍진 않습니다. **friend**, **year**, **month**, **day**, **week**, **temp_2** 변수들은 별로 도움이 되지 않는 것으로 나타났습니다. 이러한 **importances**는 모두 날씨와 관련이 없고, **week**를 최고 기온의 예측 변수로 기대하지 않기 때문에 의미가 있습니다. 또한 **year**는 모든 데이터 포인트에 대해 같으므로 최고 기온을 예측하기 위한 정보를 제공하지 않습니다.

향후 모델 구현에서는 중요하지 않은 변수를 제거할 수 있으며 성능이 저하되지 않습니다. 또한 **support vector machine**과 같은 다른 모델을 사용하는 경우, **random forest feature importances**를 일종의 **feature** 선택 방법으로 사용할 수 있습니다. 가장 중요한 두 가지 변수인 1일 전 최고 기온(**temp_1**)과 과거 평균(**average**)만 사용하여 신속하게 **random forest**를 만들고 성능을 비교해 보겠습니다.

가장 중요한 두 가지 **features** 추출

```
important_indices = [feature_list.index('temp_1'), feature_list.index('average')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]
```

Train the random forest

```
rf_most_important.fit(train_important, train_labels)
```

예측 및 오류 결정

```
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_labels)
```

성능 **metrics** 표시

```
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
mape = np.mean(100 * (errors / test_labels))
accuracy = 100 - mape
```

```
print('Accuracy:', round(accuracy, 2), '%.')
```

Mean Absolute Error: 3.9 degrees.

Accuracy: 93.8 %.

이것은 우리가 정확한 예측을 하기 위해 수집한 모든 데이터가 실제로 필요하지 않다는 것을 의미합니다! 이 모델을 계속 사용하면 두 변수만 수집하여 거의 같은 성능을 얻을 수 있습니다. 생산 환경에서는 더 많은 정보를 얻는 데 필요한 추가 시간 대비 정확도 감소를 비교해야 합니다. 성능과 비용 사이의 적절한 균형을 찾는 방법을 아는 것은 머신러닝 엔지니어에게 필수적인 기술이며 궁극적으로 문제에 달려 있습니다!

이 시점에서 우리는 **supervised regression problem**에 대한 **random forest**의 기본 구현을 위해 알아야 할 거의 모든 것을 다루었습니다. 우리 모델은 1년의 과거 데이터에서 **94%**의 정확도로 내일 최고 기온을 예측할 수 있다고 확신할 수 있습니다. 여기에서 이 예제를 자유롭게 사용하거나 원하는 **data set**에서 모델을 사용하십시오. 몇 가지 시각화를 만들어 이 게시물을 마무리하겠습니다. 데이터 과학에서 제가 가장 좋아하는 두 부분은 그래프와 모델링이므로 당연히 차트를 만들어야 합니다! 차트는 보기에 즐거울 뿐만 아니라, 많은 숫자를 빠르게 검사할 수 있는 이미지로 압축하기 때문에 모델을 진단하는 데 도움이 될 수 있습니다.

시각화

제가 만들 첫 번째 차트는 변수의 상대적 중요성의 불일치를 설명하기 위해 **feature** 중요도에 대한 간단한 막대 그림입니다. **Python**으로 **Plotting** 하는 것은 다소 직관적이지 않으며 그래프를 만들 때 **Stack Overflow**에서 거의 모든 것을 검색하게 됩니다. 여기에 있는 코드가 이해되지 않더라도 걱정하지 마세요. 원하는 최종 결과를 얻기 위해 코드를 완전히 이해하지 않아도 되는 때도 있습니다!

```
# Import matplotlib for plotting and use magic command for Jupyter Notebooks
```

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
# 스타일 설정
```

```
plt.style.use('fivethirtyeight')
```

```
# plotting 위한 x 위치 list
```

```
x_values = list(range(len(importances)))
```

```
# 막대 차트 만들기
```

```
plt.bar(x_values, importances, orientation = 'vertical')
```

```
# x 축에 대한 눈금 레이블
```

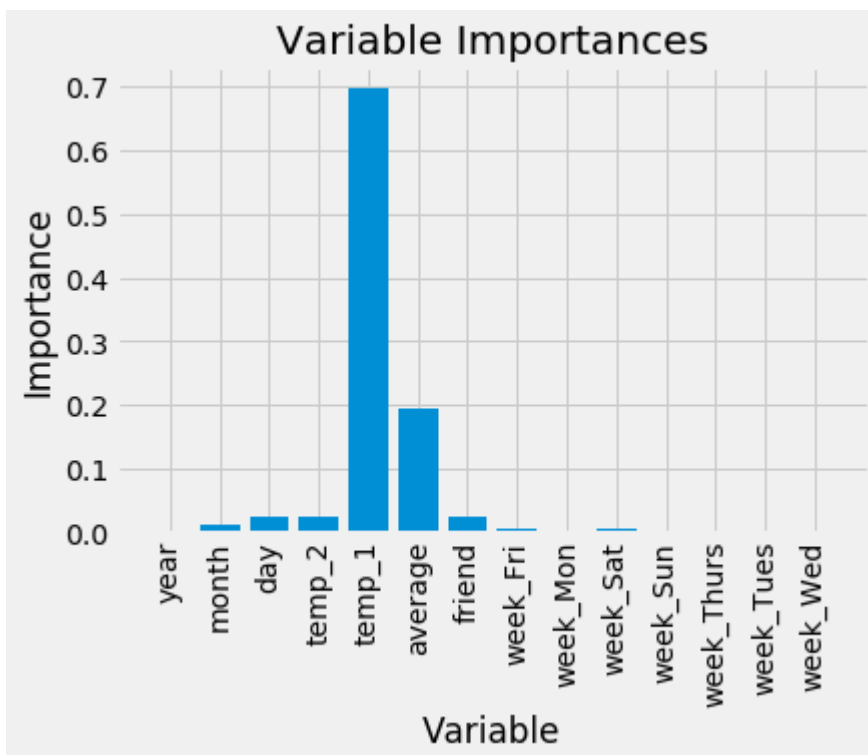
```
plt.xticks(x_values, feature_list, rotation='vertical')
```

```
# 축 레이블 및 제목
```

```
plt.ylabel('Importance')
```

```
plt.xlabel('Variable')
```

```
plt.title('Variable Importances')
```



다음으로, 예측이 강조 표시된 전체 **Dataset** 을 그릴 수 있습니다. 이것은 약간의 데이터 조작이 필요하지만 그렇게 어렵지는 않습니다. 이 **plot**을 사용하여 데이터 또는 예측에 이상치가 있는지 확인할 수 있습니다.

```
# Use datetime for creating date objects for plotting
```

```
import datetime
```

```
# 훈련 값 날짜
```

```
months = features[:, feature_list.index('month')]
```

```
days = features[:, feature_list.index('day')]
```

```
years = features[:, feature_list.index('year')]
```

```
# 나열한 다음 datetime 객체로 변환
```

```
dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year, month, day in zip(years, months, days)]
```

```
dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in dates]
```

```
# 실제 값과 날짜가있는 Dataframe
```

```
true_data = pd.DataFrame(data = {'date': dates, 'actual': labels})
```

```
# 예측 날짜
```

```
months = test_features[:, feature_list.index('month')]
```

```
days = test_features[:, feature_list.index('day')]
```

```
years = test_features[:, feature_list.index('year')]
```

```
# 날짜 열
```

```
test_dates = [str(int(year)) + '-' + str(int(month)) + '-' + str(int(day)) for year, month, day in zip(years, months, days)]
```

```
# datetime 객체로 변환
```

```
test_dates = [datetime.datetime.strptime(date, '%Y-%m-%d') for date in test_dates]
```

```
# Dataframe with predictions and dates
```

```
predictions_data = pd.DataFrame(data = {'date': test_dates, 'prediction': predictions})
```

```
# Plot the actual values
```

```
plt.plot(true_data['date'], true_data['actual'], 'b-', label = 'actual')
```

```
# Plot the predicted values
```

```
plt.plot(predictions_data['date'], predictions_data['prediction'], 'ro', label = 'prediction')
```

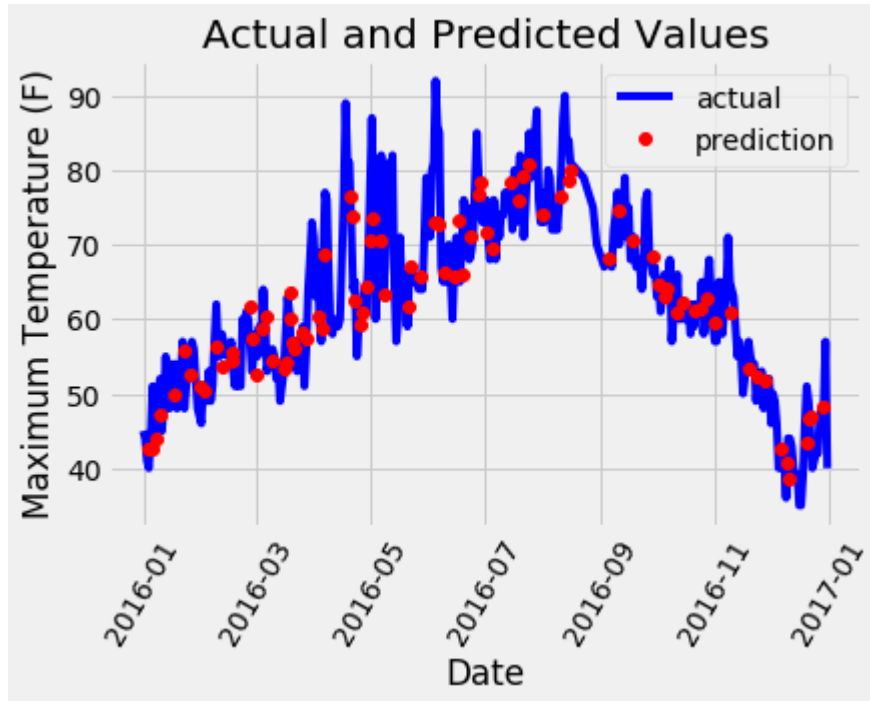
```
plt.xticks(rotation = '60')
```

```
plt.legend()
```

```
# Graph labels
```

```
plt.xlabel('Date'); plt.ylabel('Maximum Temperature (F)')
```

```
plt.title('Actual and Predicted Values')
```



멋진 그래프를 만들기 위한 약간의 작업! 수정해야 할 눈에 띄는 이상치는 보이지 않습니다. 모델을 추가로 진단하기 위해 잔차(오류)를 플로팅하여 모델이 과도하게 예측하거나 과소 예측하는 경향이 있는지 확인할 수 있으며 잔차가 정규 분포를 따르는지도 확인할 수 있습니다. 그러나 실제 값, 하루 전의 기온, 과거 평균 및 친구의 예측을 보여주는 최종 차트를 하나 만들 것입니다. 이를 통해 유용한 변수와 그다지 도움 되지 않는 변수의 차이를 확인할 수 있습니다.

```
# Make the data accessible for plotting
```

```
true_data['temp_1'] = features[:, feature_list.index('temp_1')]
```

```
true_data['average'] = features[:, feature_list.index('average')]
```

```
true_data['friend'] = features[:, feature_list.index('friend')]
```

```
# 모든 데이터를 선으로 Plot
```

```
plt.plot(true_data['date'], true_data['actual'], 'b-', label = 'actual', alpha = 1.0)
```

```
plt.plot(true_data['date'], true_data['temp_1'], 'y-', label = 'temp_1', alpha = 1.0)
```

```
plt.plot(true_data['date'], true_data['average'], 'k-', label = 'average', alpha = 0.8)
```

```
plt.plot(true_data['date'], true_data['friend'], 'r-', label = 'friend', alpha = 0.3)
```



```
# Formatting plot
```

```
plt.legend()
```

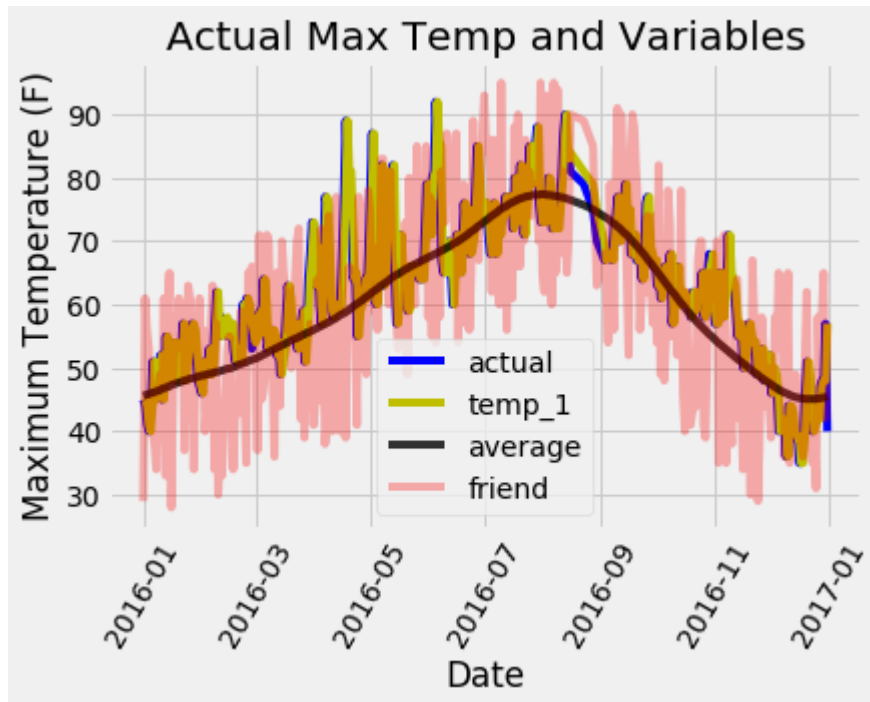
```
plt.xticks(rotation = '60')
```

```
# Labels and title
```

```
plt.xlabel('Date')
```

```
plt.ylabel('Maximum Temperature (F)')
```

```
plt.title('Actual Max Temp and Variables')
```



모든 선을 확인하는 것은 약간 어렵지만, 하루 전의 최고 기온(temp_1)과 과거 최고 기온(average)이 최고 기온을 예측하는 데 유용한 이유를 알 수 있습니다. (아직 friend를 포기하지 마세요. 추정치에 너무 많은 비중을 두지 않을 수도 있습니다!) 이와 같은 그래프는 변수를 선택할 수 있도록 도움이 되는 경우가 많습니다. 또한 진단에도 사용할 수 있습니다. Anscombe's quartet의 경우와 마찬가지로 그래프는 종종 양적 수치보다 더 많이 드러나고 머신러닝 workflow 일부여야 합니다.

결론

이러한 그래프를 사용하여 전체 **end-to-end** 머신 러닝 예제를 완료했습니다! 이 시점에서 모델을 개선하려면 다른 **hyperparameters**(설정)를 사용해 다른 알고리즘을 시도하거나 가장 좋은 방법은 더 많은 데이터를 수집하는 것입니다! 모든 모델의 성능은 학습할 수 있는 유효한 데이터의 양에 정비례하고, 우리는 **training**에 매우 제한된 양의 정보를 사용했습니다. 저는 누구나 이 모델을 시도하고 개선하고 결과를 공유하도록 권장합니다. 여기에서 수많은 [온라인 \(무료\) 리소스](#)를 사용하여 [랜덤 포레스트 이론](#)과 응용 프로그램에 대해 자세히 알아볼 수 있습니다.

[출처 :

<https://towardsdatascience.com/random-forest-in-python-24d0893d51c0>]