Model Interpretability

LIME—python

Installation:

```
koushunantekiMacBook-Pro:~ mamamia$ pip install lime
Requirement already satisfied: lime in ./anaconda3/lib/python3.6/site-packages (
0.1.1.32)
Requirement already satisfied: scikit-learn>=0.18 in ./anaconda3/lib/python3.6/s
ite-packages (from lime) (0.20.0)
Requirement already satisfied: scikit-image>=0.12 in ./anaconda3/lib/python3.6/s
ite-packages (from lime) (0.13.1)
Requirement already satisfied: scipy in ./anaconda3/lib/python3.6/site-packages
(from lime) (1.1.0)
```

The lime package is on PyPI, Simply run in command line:

Text classifier demo on Jupyter notebook:

1. Import library:

```python
import lime
import sklearn
import numpy as np
import sklearn
import sklearn.ensemble
import sklearn.metrics
from __future__ import print_function
```

2. Fetching data, train a classifier:

For this demo, we'll be using the [20 newsgroups dataset](). In particular, for simplicity, we'll use a 2-class subset:

```python
from sklearn.datasets import fetch_20newsgroups
categories = ['alt.atheism', 'soc.religion.christian']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
class_names = ['atheism', 'christian']
```

atheism and christianity.

3.

```
vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(lowercase=False)
train_vectors = vectorizer.fit_transform(newsgroups_train.data)
test_vectors = vectorizer.transform(newsgroups_test.data)
```

Vectorize the text data:
Let's use the tfidf vectorizer, commonly used for text.

4.

```
rf = sklearn.ensemble.RandomForestClassifier(n_estimators=500)
rf.fit(train_vectors, newsgroups_train.target)
```

RandomForestClassifier:

Explaining prediction using LIME:

Lime explainers assume that classifiers act on raw text, but sklearn classifiers act on vectorized representation of texts.

1. Use sklearn's pipeline, and implements predict_proba on raw_text lists.

```
from lime import lime_text
from sklearn.pipeline import make_pipeline
c = make_pipeline(vectorizer, rf)
```

```
print(c.predict_proba([newsgroups_test.data[0]]))
```

```
[[0.286 0.714]]
```

2. Create an explainer object

```
from lime.lime_text import LimeTextExplainer
explainer = LimeTextExplainer(class_names=class_names)
```

```
idx = 83
exp = explainer.explain_instance(newsgroups_test.data[idx], c.predict_proba, num_features=6)
print('Document id: %d' % idx)
print('Probability(christian) =', c.predict_proba([newsgroups_test.data[idx]])[0,1])
print('True class: %s' % class_names[newsgroups_test.target[idx]])
```

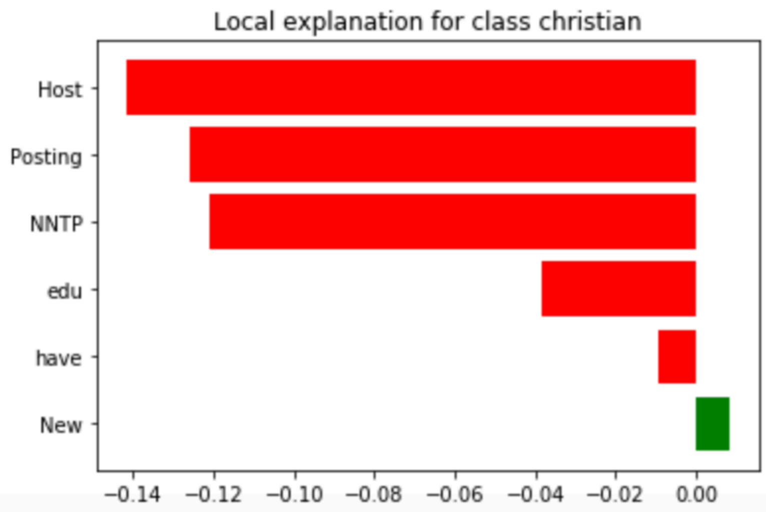
3. Generate an explanation with at most 6 features for an arbitrary document in the test set

```
Document id: 83
Probability(christian) = 0.452
True class: atheism
```

Visualizing explanation:

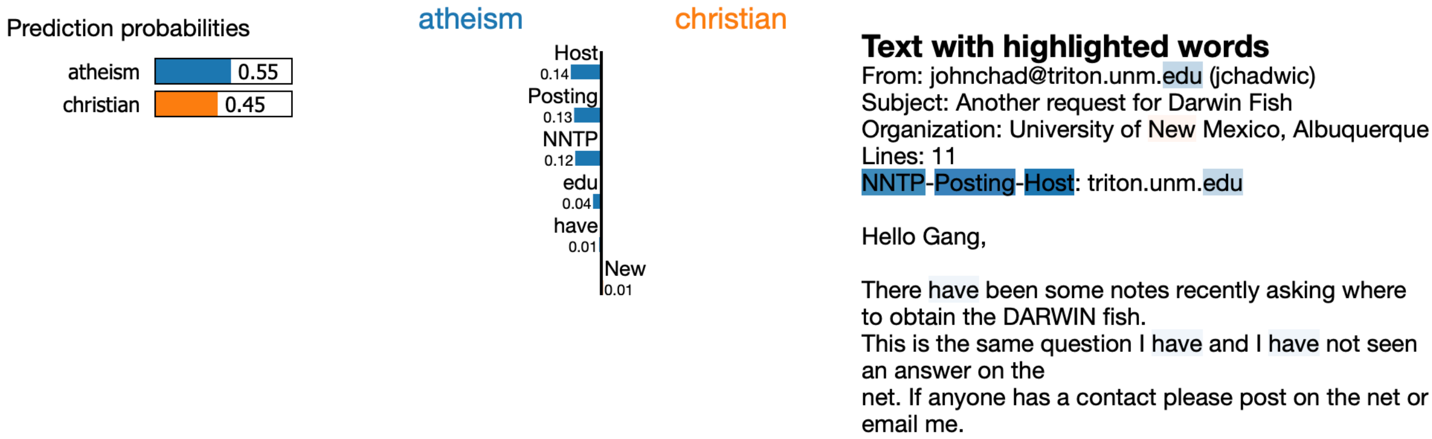
The explanations can be returned as a matplotlib barplot:

```
%matplotlib inline
fig = exp.as_pyplot_figure()
```



Finally, we can also include a visualization of the original document, with the words in the explanations highlighted. Notice how the words that affect the classifier the most are all in the email header.

```
exp.show_in_notebook(text=True)
```



iml — R package

Installation:

1. If you want to use jupyter notebook to edit. You must install the R essentials in your current environment:

Run this in command line:

```
conda install -c r r-essentials
```

2. After That you can open a jupyter notebook with R kernel and then install iml package. If you want to use jupyter notebook you must install the package under the anaconda3 R library.

Run this in R console:

```
install.packages("iml", "/home/user/anaconda3/lib/R/library")
```

Boston housing price explanation demo:

We'll use the MASS::Boston dataset to demonstrate the abilities of the iml package. This dataset contains median house values from Boston neighborhoods.

```
# Loading the packages
library(iml)
# We use the mlr package for training the machine learning models
library("randomForest")
```

```
data("Boston", package  = "MASS")
head(Boston)
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4

Train a randomForest to predict the Boston median housing value:

```
rf = randomForest(medv ~ ., data = Boston, ntree = 50)
```

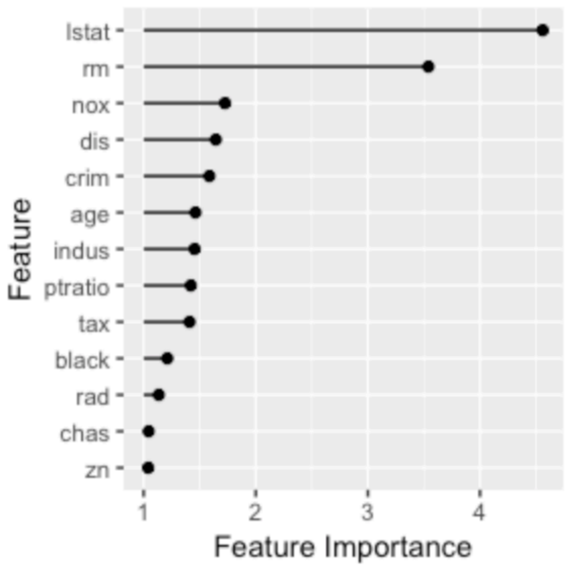
Create a Predictor object, that holds the model and the data:

```
X = Boston[which(names(Boston) != "medv")]
predictor = Predictor$new(rf, data = X, y = Boston$medv)
```

Feature Importance:

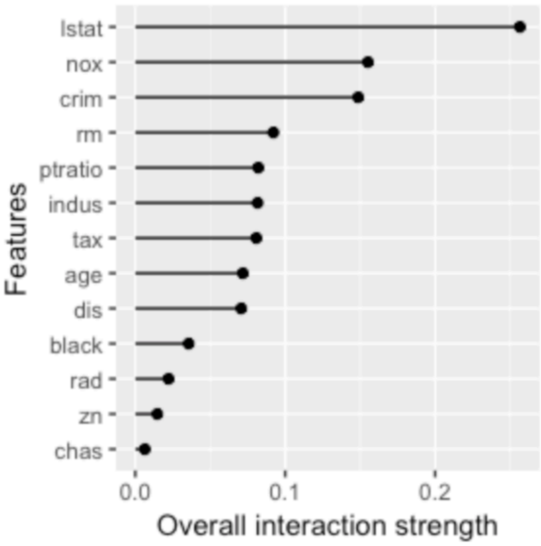
We can measure how important each feature was for the predictions with FeatureImp

```
library(repr)
options(repr.plot.width=7, repr.plot.height=6)
imp = FeatureImp$new(predictor, loss = "mae")
plot(imp)
```



Feature Interactions:

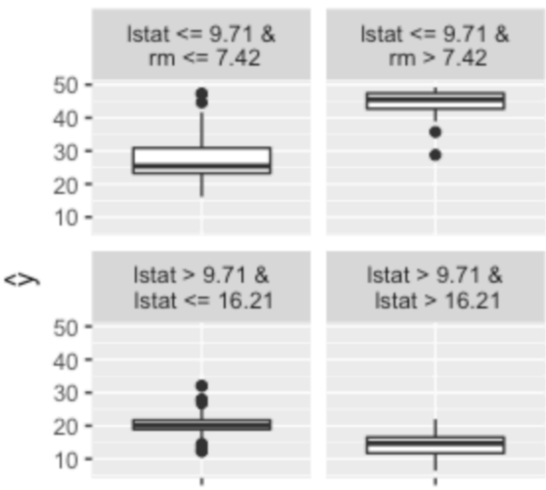
```
interact = Interaction$new(predictor)
plot(interact)
```



Surrogate Model:

Replace the black box with a simpler model - a decision tree. We take the predictions of the black box model (in our case the random forest) and train a decision tree on the original features and the predicted outcome.

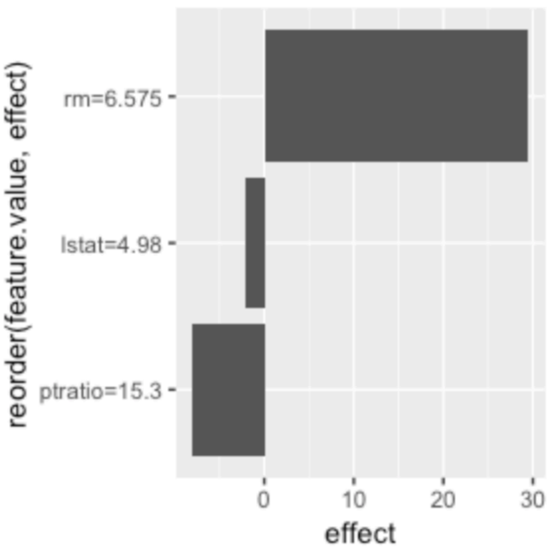
```
tree = TreeSurrogate$new(predictor, maxdepth = 2)
plot(tree)
```



Explain single prediction with a local model:

Fit a model locally to understand an individual prediction better. The local model fitted by `LocalModel` is a linear regression model and the data points are weighted by how close they are to the data point for which we want to explain the prediction.

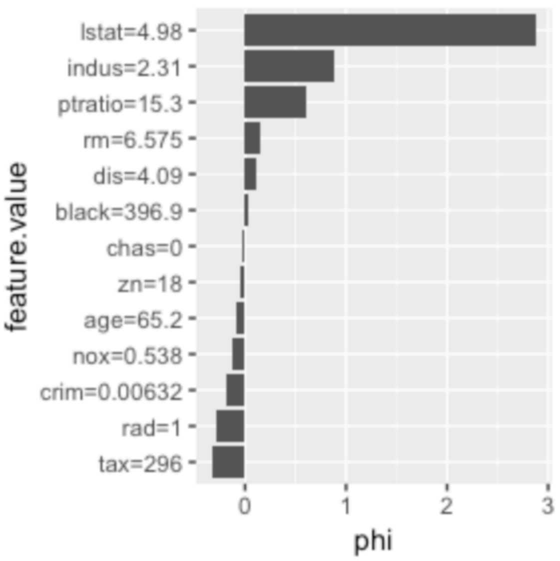
```
lime.explain = LocalModel$new(predictor, x.interest = X[1,])
lime.explain$results
plot(lime.explain)
```



Explain single prediction with game theory:

Assume that for one data point, the feature values play a game together, in which they get the prediction as a payout. The Shapley value tells us how to fairly distribute the payout among the feature values.

```
shapley = Shapley$new(predictor, x.interest = X[1,])
shapley$plot()
```



Reference:

PPT:

Understanding model predictions with LIME
https://towardsdatascience.com/understanding-model-predictions-with-lime-a582fdff3a3b

LIME - Local Interpretable Model-Agnostic Explanations
https://homes.cs.washington.edu/~marcotcr/blog/lime/

Introduction to Local Interpretable Model-Agnostic Explanations (LIME)
https://www.oreilly.com/learning/introduction-to-local-interpretable-model-agnostic-explanations-lime

Prediction Explanation: Adding Transparency to Machine Learning
https://blog.bigml.com/2018/05/01/prediction-explanation-adding-transparency-to-machine-learning/

Interpretable Machine Learning Using LIME Framework - Kasia Kulma (PhD), Data Scientist, Aviva
https://www.slideshare.net/0xdata/interpretable-machine-learning-using-lime-framework-kasia-kulma-phd-data-scientist

Interpretable Machine Learning-A Guide for Making Black Box Models Explainable. Christoph Molnar
https://christophm.github.io/interpretable-ml-book/

LIME—Python:

https://github.com/marcotcr/lime/tree/master/doc/notebooks

iml - R package:

Chapter 5 Model-Agnostic Methods
https://christophm.github.io/interpretable-ml-book/agnostic.html

Introduction to iml: Interpretable Machine Learning in R
https://cran.r-project.org/web/packages/iml/vignettes/intro.html

Skater—python

Installation:

In order to access the full function of Skater, please check these pre-requisites installed properly

Dependencies

Skater relies on

- scikit-learn>=0.18,
- pandas>=0.19,
- ds-lime>=0.1.1.21(datascience.com forked version of LIME),
- requests,
- multiprocess,
- joblib==0.11,
- dill>=0.2.6,
- rpy2==2.9.1; python_version>"3.0",
- numpy
- with v1.1.0 there are additional dependencies on R related binaries(setup.sh)

External/Optional dependencies

- Plotting functionality requires matplotlib>=2.1.0
- tensorflow>=1.4.0
- keras>=2.0.8

in the environment.
Run the following command line for the latest version (Nov 2018):

```
pip install -U skater==1.1.2
```

```
Requirement already satisfied, skipping upgrade: decorator>=4.3.0 in /anaconda3/
lib/python3.6/site-packages (from networkx>=1.8->scikit-image==0.14->skater) (4.
3.0)
Requirement already satisfied, skipping upgrade: toolz>=0.7.3; extra == "array"
in /anaconda3/lib/python3.6/site-packages (from dask[array]>=0.9.0->scikit-image
==0.14->skater) (0.9.0)
Requirement already satisfied, skipping upgrade: setuptools in /anaconda3/lib/py
thon3.6/site-packages (from kiwisolver>=1.0.1->matplotlib>=2.0.0->scikit-image==
0.14->skater) (40.4.3)
OsamuyadeMacBook-Pro:~ osamuyanagano$ 
```

……

Preparation:

```python
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import pandas as pd
# Reference for customizing matplotlib: https://matplotlib.org/users/style_sheets.html
plt.style.use('ggplot')

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from skater.core.explanations import Interpretation
from skater.model import InMemoryModel


data = load_breast_cancer()
# Description of the data
print(data.DESCR)

pd.DataFrame(data.target_names)
```

1.Please use the following block of code to import necessary modules.
In this demo, we use a breast-cancer dataset to perform predictions. Its features are all linear and the target is shown as the following:

Out[1]:

	0
0	malignant
1	benign

```
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2)

clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()

eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='soft')
eclf = eclf.fit(X_train, y_train)

clf1 = clf1.fit(X_train, y_train)
clf2 = clf2.fit(X_train, y_train)
clf3 = clf3.fit(X_train, y_train)
```

2.Create a complex model to be interpreted:
The three simple classifiers ensembles a complex classifier “eclf”.

```
interpreter = Interpretation(X_test, feature_names=data.feature_names)
```

3. Before any interpretation is made, we should initialize an interpretation object for the model:

Global Interpretation—Feature Importance:

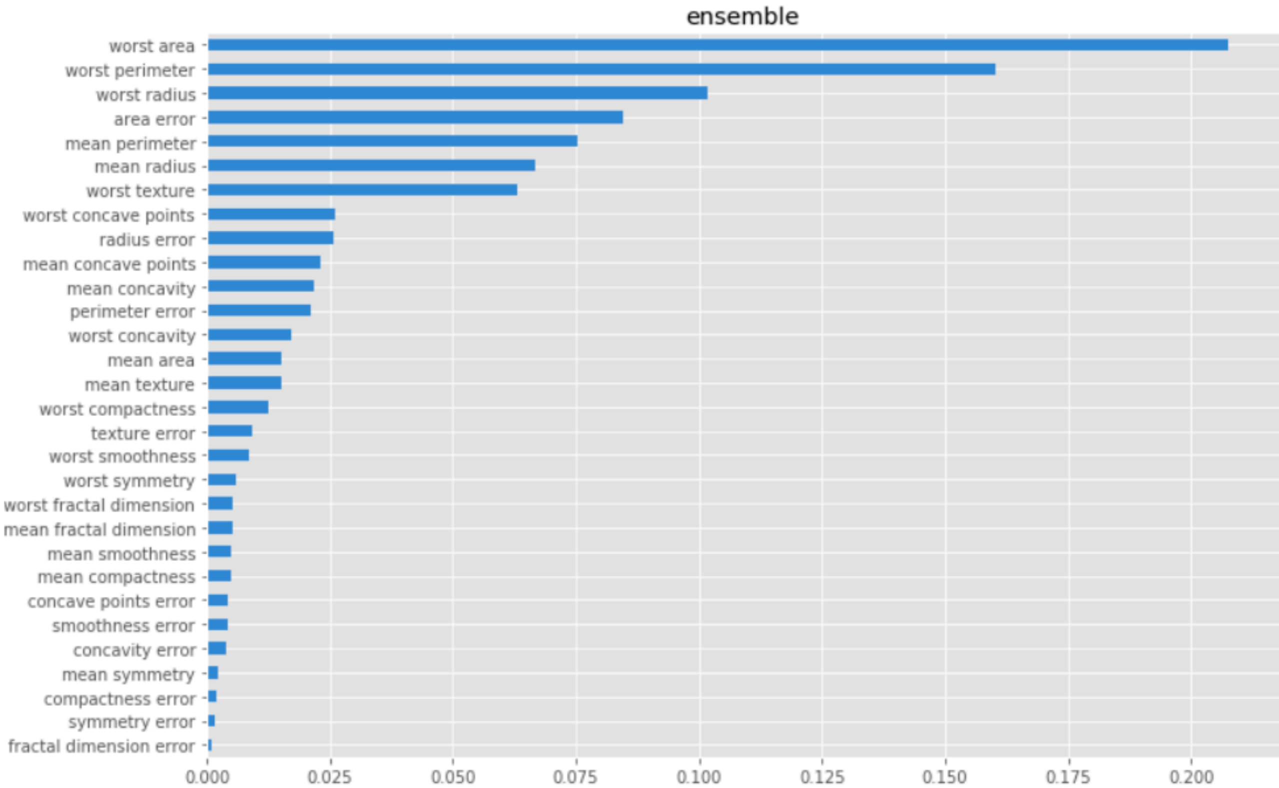
Skater provides a feature importance plotting method eligible for any model. To use that:

1. Create an InMemoryModel for your ensemble model.

2. Call the functions of interpreter accordingly and pass this model object as its parameter.

```
# Ensemble Classifier does not have feature importance enabled by default
pyint_model = InMemoryModel(eclf.predict_proba, examples=X_test)
interpreter.feature_importance.plot_feature_importance(pyint_model, ascending=True)
```

For example:



And we have:

Global Interpretation — Partial Dependency:

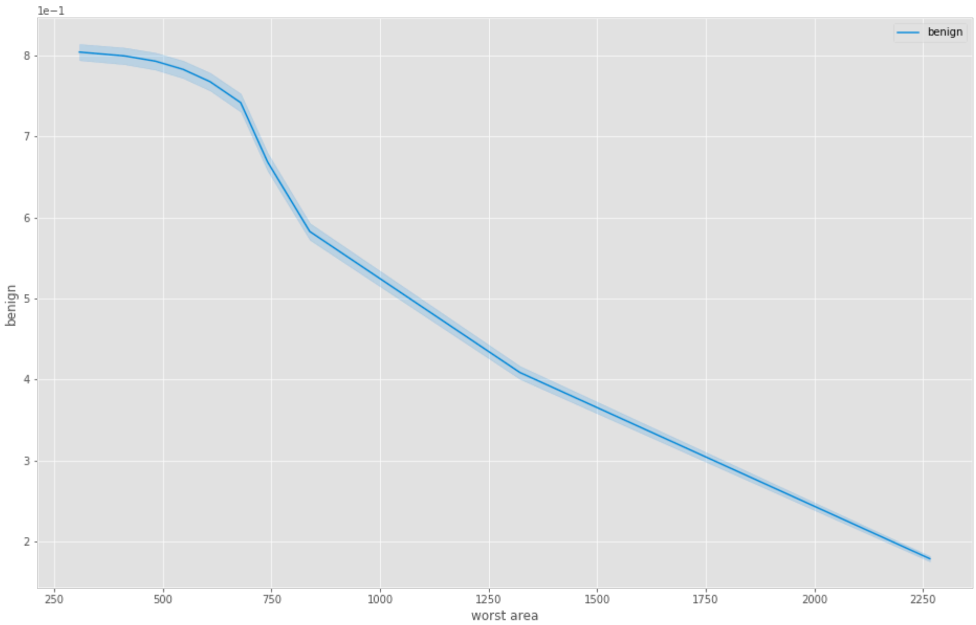
Skater also provides a special visualization function to explain the partial dependency of the features, that is, in the current model, how do the feature affect the target.

1. To explain the correlation between the target and a single feature. We can pass multiple features in a list to a single call of function, and the function will plot a figure for each feature

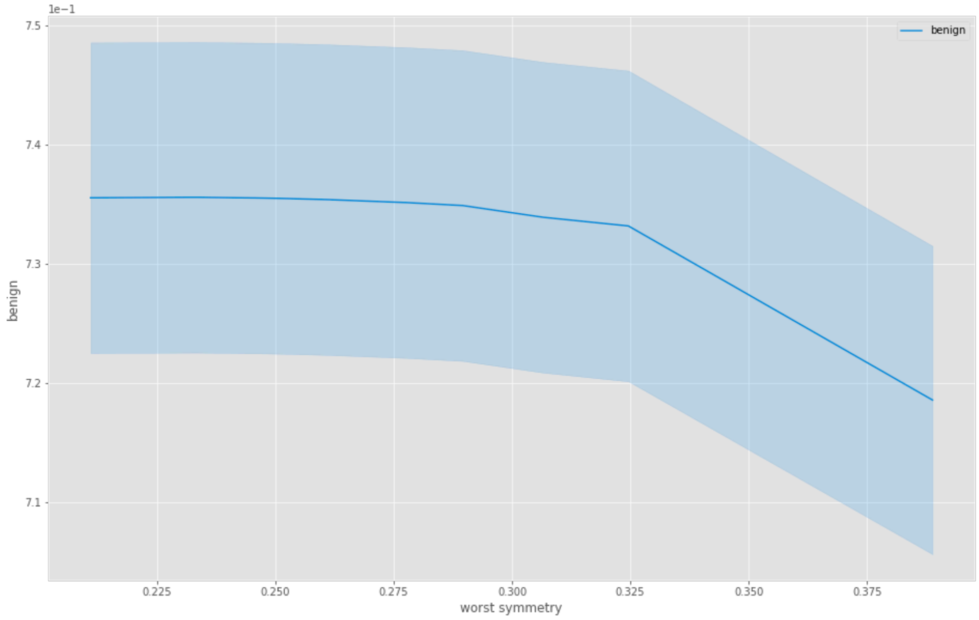
```
interpreter.partial_dependence.plot_partial_dependence(list(['worst area', 'worst symmetry']),
                                                       pyint_model,
                                                       grid_resolution=10,
                                                       with_variance=True)
```

respectively.

Please note that the ‘grid_resolution’ parameter here is inherited from sklearn, stands for the number of equally spaced points on the axes. Large values indicate preciseness and lack of



smoothness on the figure.

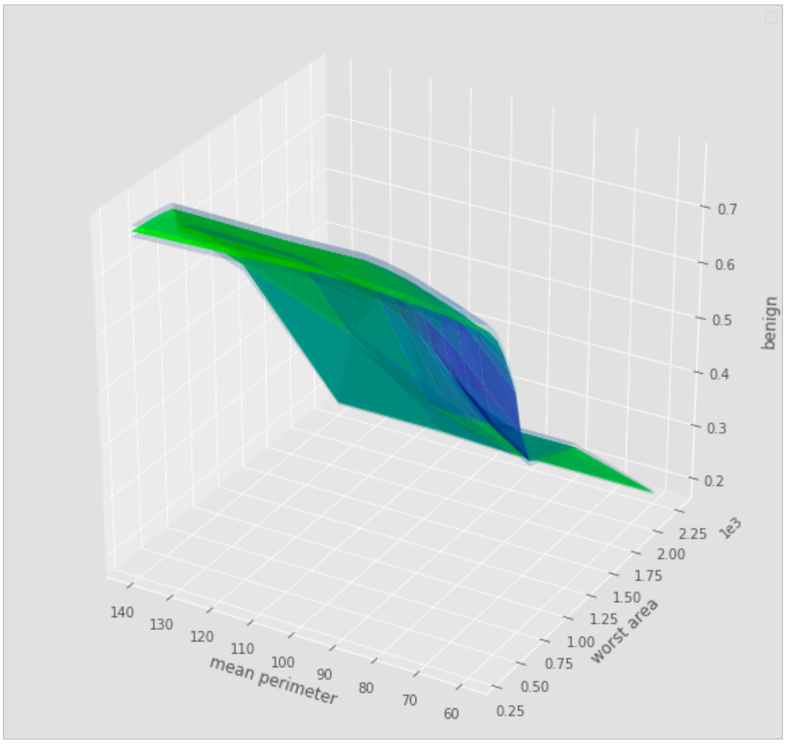


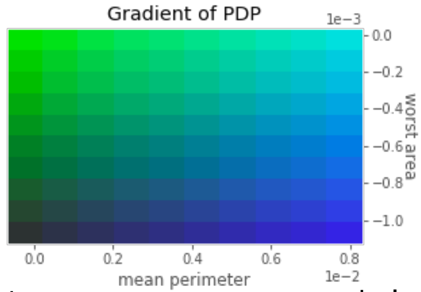
The colored margins in the figures stands for the variance of predicted values. High variance means weak correlation, and vice versa.

```
axes_list = interpreter.partial_dependence.plot_partial_dependence([['mean perimeter','worst area']],
                                                                    pyint_model,
                                                                    grid_resolution=grid_resolution.value,
                                                                    with_variance=True)
```

2. The function can be overloaded by a tow-dimensional array contains a pair of features.

In this case, the function plots two figures explaining the dependency between the target and the two features as a whole.





‘PDP’: Partial Dependency Plot

Local Interpretation — Using LIME In Skater:

Skater also integrated LIME explanations to perform local interpretation. To use that:

```
from skater.core.local_interpretation.lime.lime_tabular import LimeTabularExplainer
from IPython.display import display, HTML, clear_output
```

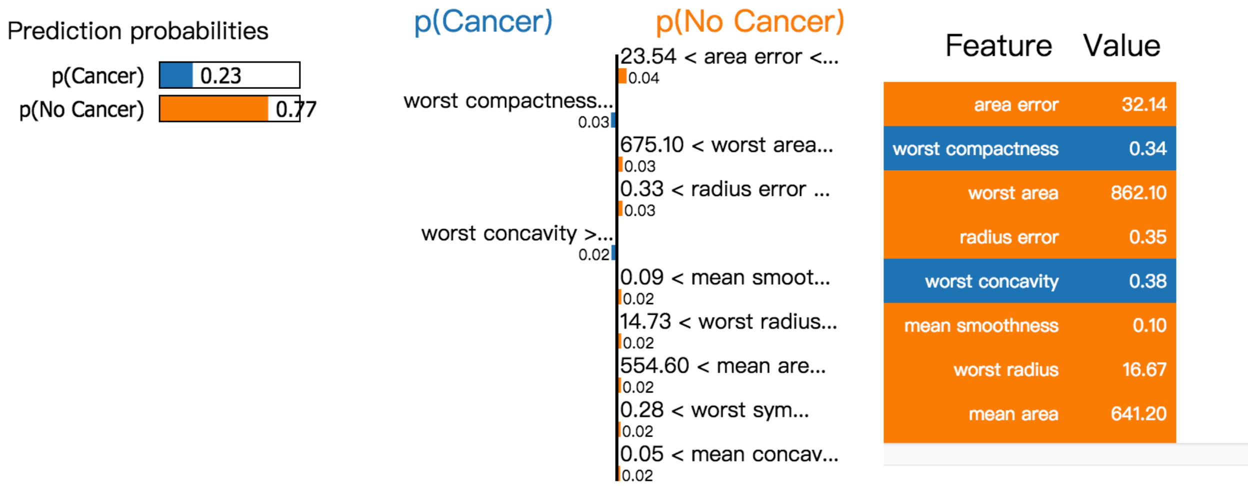
1. Import necessary packages.

```
exp = LimeTabularExplainer(X_test,
                           feature_names=data.feature_names,
                           class_names=['p(Cancer)', 'p(No Cancer)'])
```

2.Create a LIME object.
Feed the object “class_names” to label its predictions.

```
print("Model behavior at row: {}".format(2))
# Lets evaluate the prediction from the model and actual target label
print("prediction from the model:{}".format(eclf.predict(X_test[2].reshape(1, -1))))
print("Target Label on the row: {}".format(y_test.reshape(1,-1)[0][2]))
clear_output()
display(HTML(exp.explain_instance(X_test[2], eclf.predict_proba).as_html()))
```

3. Display the 2nd prediction, using the ensemble classifier ‘eclf’.



In comparison with LIME framework, the results shows:

Reference:

https://datascienceinc.github.io/Skater/install.html

https://datascienceinc.github.io/Skater/gallery.html#interpretation-examples

https://github.com/datascienceinc/Skater

https://www.oreilly.com/ideas/interpreting-predictive-models-with-skater-unboxing-model-opacity

Reproducibility

To Create Your First PyPI Package

High Level Steps

Idea Made to Local Package

Version Control of Project (Git)

Upload to PyPi

The Tree of Directory of Final

1.

```
Ying:PackageDemo wangying$ tree
.
├── LICENSE
├── Makefile
├── README.md
├── build
│   ├── bdist.macosx-10.7-x86_64
│   └── lib
│       └── yingpackage
│           ├── __init__.py
│           └── ying.py
├── dist
│   ├── yingpackage-0.0.1-py3-none-any.whl
│   └── yingpackage-0.0.1.tar.gz
├── setup.py
├── yingpackage
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-37.pyc
│   │   └── ying.cpython-37.pyc
│   ├── tests
│   │   ├── __pycache__
│   │   │   └── test_ying.cpython-37-PYTEST.pyc
│   │   ├── test.py
│   │   └── test_ying.py
│   └── ying.py
└── yingpackage.egg-info
    ├── PKG-INFO
    ├── SOURCES.txt
    ├── dependency_links.txt
    └── top_level.txt
```

1) Code Made to Local Package – build local package

1. The Code to, create a ying.py

```
def do():
        return 'testing package!'
```

2. Make directory to package, create a directory called yingpackage
   Create a file call __init__.py inside

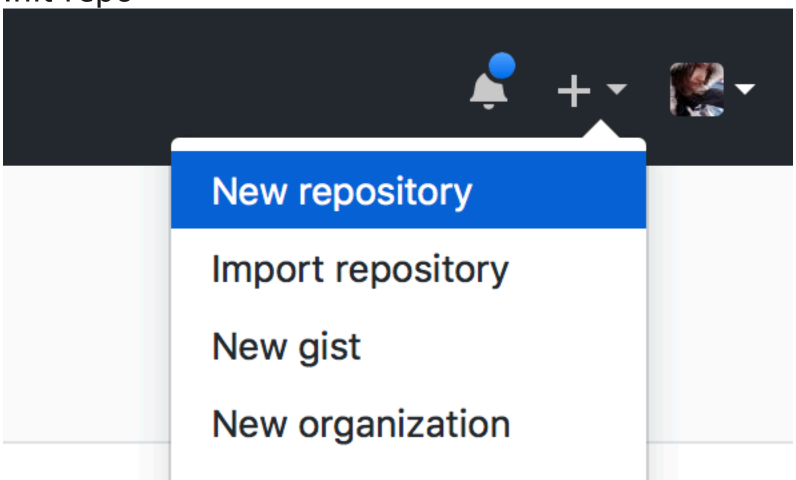
```
Ying:PackageDemo wangying$ tree
.
└── yingpackage
    ├── __init__.py
    └── ying.py
```

3. ipython to test the local package

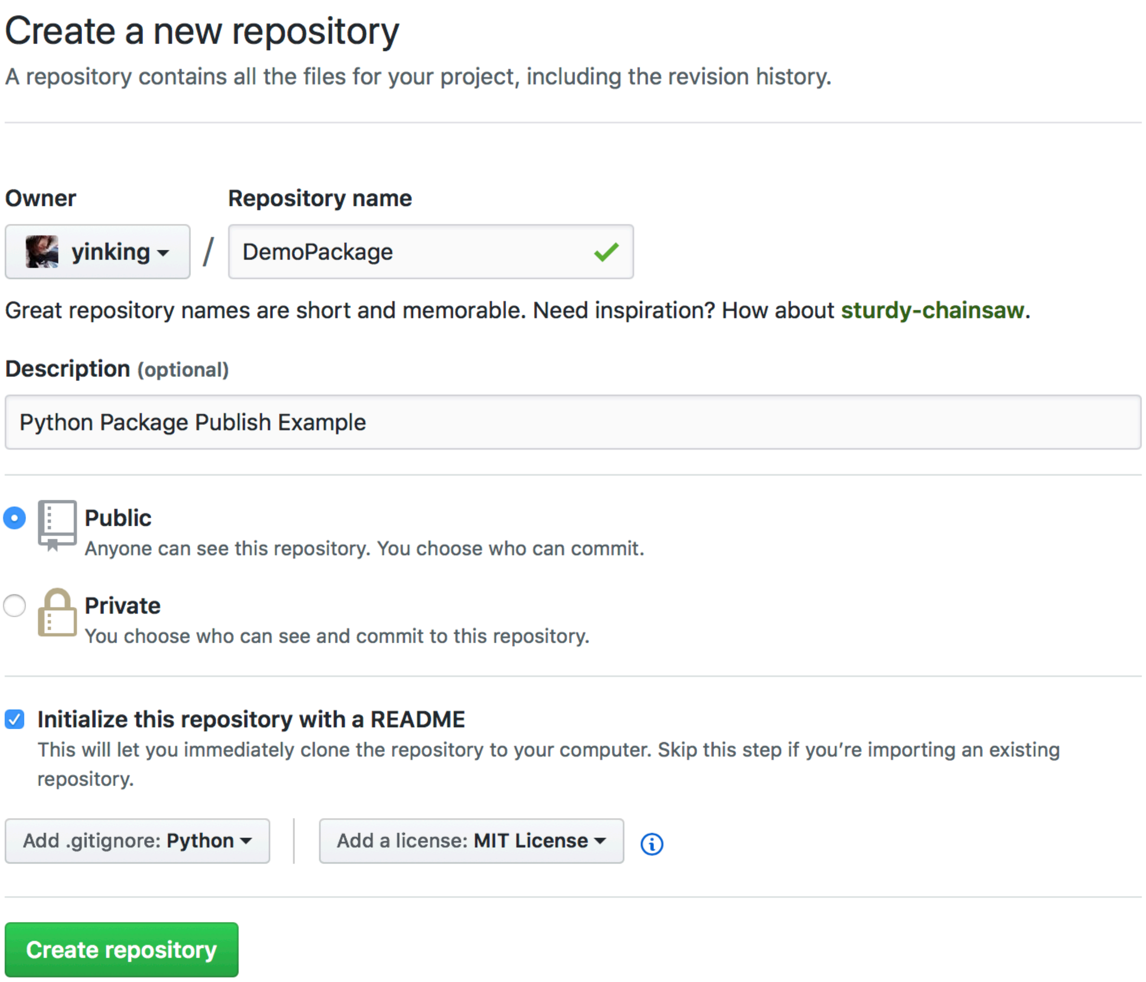
```
Ying:PackageDemo wangying$ ipython
Python 3.7.0 (default, Jun 28 2018, 07:39:16)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from yingpackage.ying import do

In [2]: do()
Out[2]: 'testing package!'

In [3]:
```

2) Version of our Project (The link to my repo https://github.com/yinking/PackageDemo)

1. Init repo



2.



3) Upload to PyPI

1. Creating setup.py

```
import setuptools

setuptools.setup(
    name="yingpackage",
    version="0.0.1",
    author="Ying Wang",
    author_email="gladyswang422@gmail.com",
    description="7390 PackageDemo",
    url="https://github.com/yinking/PackageDemo",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
```

```
    packages=setuptools.find_packages(exclude=['yingpackage.tests'])
)
```

2. Generating distribution archives

Register to PyPi: https://test.pypi.org/account/register/

Make sure you have the latest versions of setuptools and wheel installed:

```
python3 -m pip install --user --upgrade setuptools wheel
```

Once installed, run Twine to upload all of the archives under dist:

```
twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

You will be prompted for the username and password you registered with Test PyPI. After the command completes, you should see output similar to this:

```
Uploading distributions to https://test.pypi.org/legacy/
Enter your username: [your username]
Enter your password:
Uploading example_pkg-0.0.1-py3-none-any.whl
100%|█████████████████████| 4.65k/4.65k [00:01<00:00, 2.88kB/s]
Uploading example_pkg-0.0.1.tar.gz
100%|█████████████████████| 4.25k/4.25k [00:01<00:00, 3.05kB/s]
```

4) Installing your newly uploaded package¶

You can use pip to install your package and verify that it works. Create a new virtualenv (see Installing Packages for detailed instructions) and install your package from TestPyPI:

```
pip install -i https://test.pypi.org/simple/ yingpackageNote
```

If you used a different package name in the preview step, replace `example_pkg` in the command above with your package name.

```
ipython
```

And then import the module and print out the `name` property. This should be the same regardless of what you name you gave your distribution package in `setup.py`

```
>>> from yingpackage.ying import do
>>> do()
```

```
Demo PyPi repo: https://test.pypi.org/project/yingpackage/
Demo Git repo : https://github.com/yinking/PackageDemo
```

Summary (A more complete reproducibility)

- Parameterize your notebooks: How to pass in parameters to notebooks
- Test your notebooks: How to validate your notebooks
- Deploy your notebooks: How to share your notebooks
- Typeset equations
- CI (continuous integration)
- Documentation
- Version control
- Containerize