Python Libraries for Interpretable Machine Learning

Sep 2019 Gold KDnuggets Blog

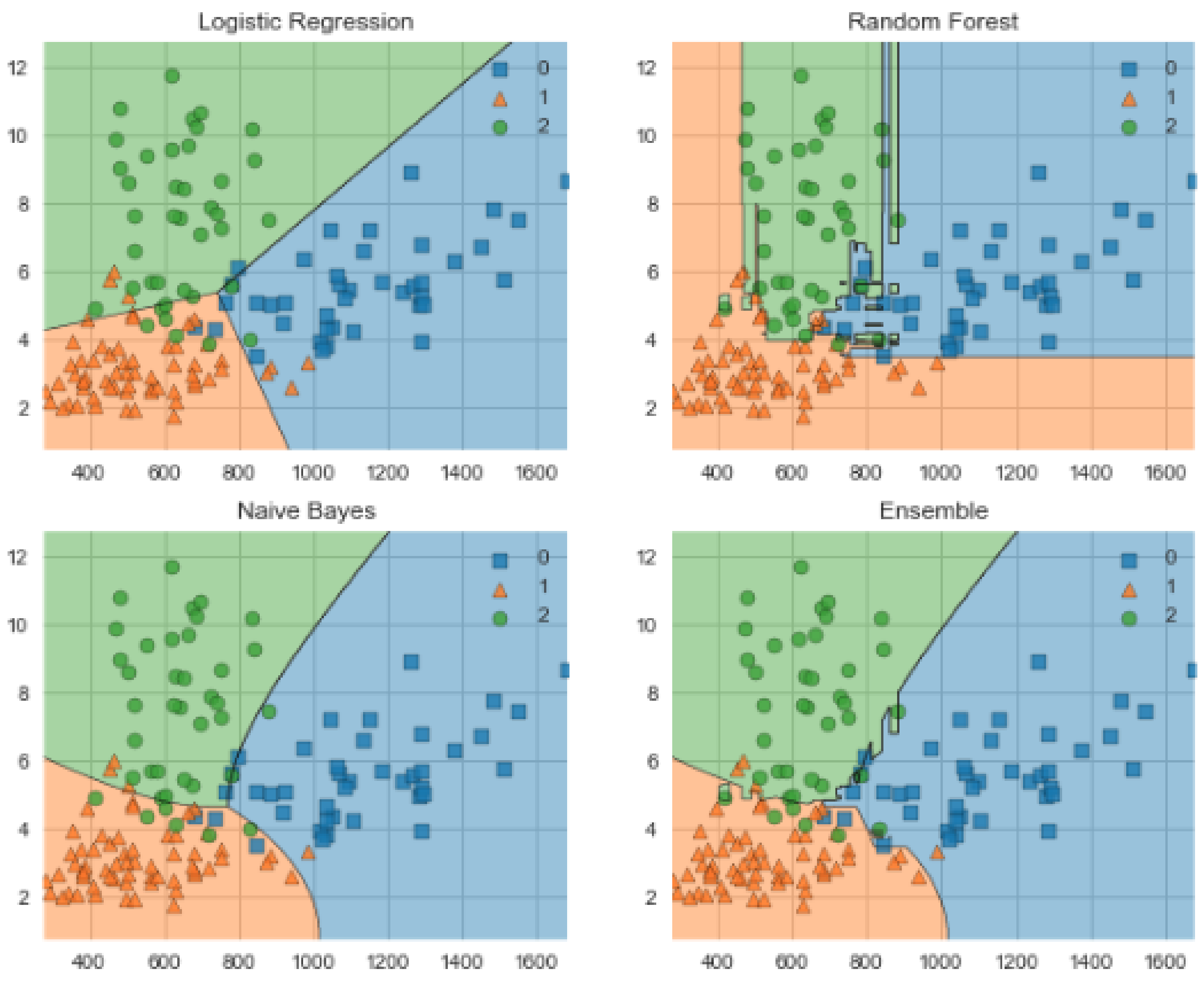
<= Previous post
Next post =>

Tags: Bias, Interpretability, LIME, Machine Learning, Python, SHAP

In the following post, I am going to give a brief guide to four of the most established packages for interpreting and explaining machine learning models.

comments

By Rebecca Vickery, Data Scientist



As concerns regarding bias in artificial intelligence become more prominent it is becoming more and more important for businesses to be able to explain both the predictions their models are producing and how the models themselves work. Fortunately, there is an increasing number of python libraries being developed that attempt to solve this problem. In the following post, I am going to give a brief guide to four of the most established packages for interpreting and explaining machine learning models.

The following libraries are all pip installable, come with good documentation and have an emphasis on visual interpretation.

yellowbrick

This library is essentially an extension of the scikit-learn library and provides some really useful and pretty looking visualisations for machine learning models. The visualiser objects, the core interface, are scikit-learn estimators and so if you are used to working with scikit-learn the workflow should be quite familiar.

The visualisations that can be rendered cover model selection, feature importances and model performance analysis.

Let’s walk through a few brief examples.

The library can be installed via pip.

```
pip install yellowbrick
```

To illustrate a few features I am going to be using a scikit-learn dataset called the wine recognition set. This dataset has 13 features and 3 target classes and can be loaded directly from the scikit-learn library. In the below code I am importing the dataset and converting it to a data frame. The data can be used in a classifier without any additional preprocessing.

```
import pandas as pd
from sklearn import datasets
```

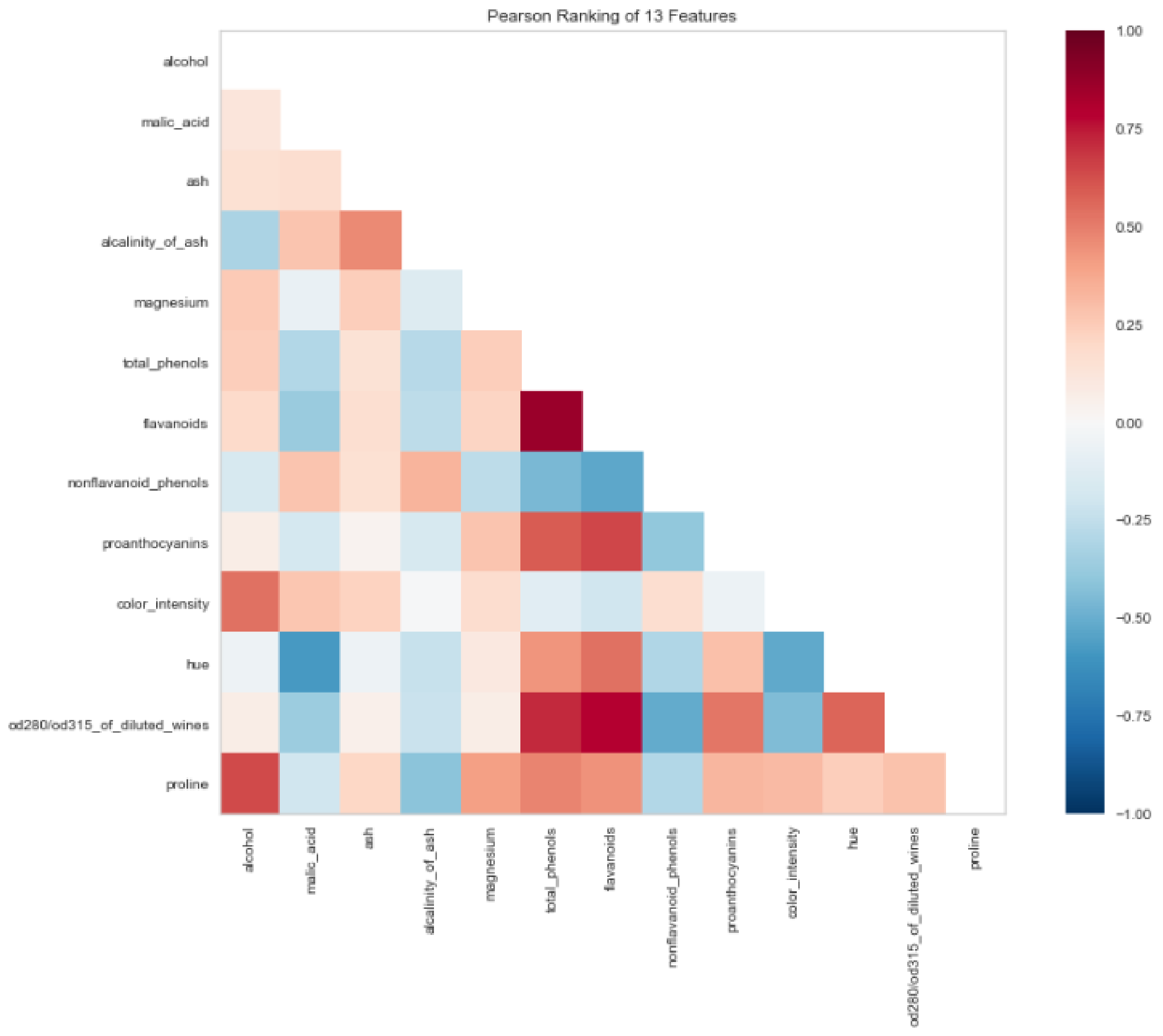
```
wine_data = datasets.load_wine()
df_wine = pd.DataFrame(wine_data.data,columns=wine_data.feature_names)
df_wine['target'] = pd.Series(wine_data.target)
```

I am also using scikit-learn to further split the data set into test and train.

```
from sklearn.model_selection import train_test_split
X = df_wine.drop(['target'], axis=1)
y = df_wine['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

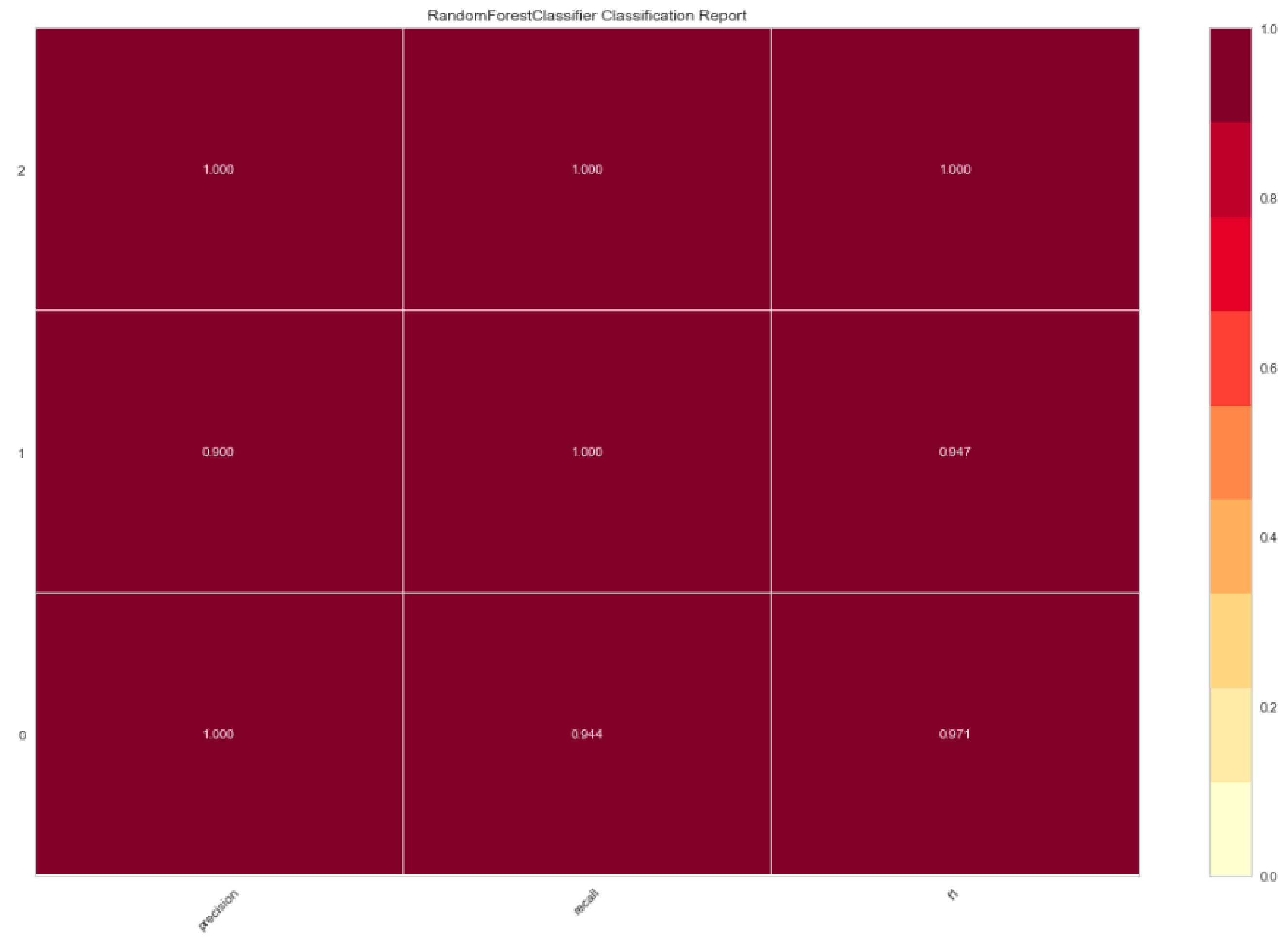
Next, let's use the Yellowbricks visualiser to view correlations between features in the data set.

```
from yellowbrick.features import Rank2D
import matplotlib.pyplot as plt
visualizer = Rank2D(algorithm="pearson",  size=(1080, 720))
visualizer.fit_transform(X_train)
visualizer.poof()
```



Let's now fit a RandomForestClassifier and evaluate the performance with another visualiser.

```
from yellowbrick.classifier import ClassificationReport
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
visualizer = ClassificationReport(model, size=(1080, 720))
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.poof()
```

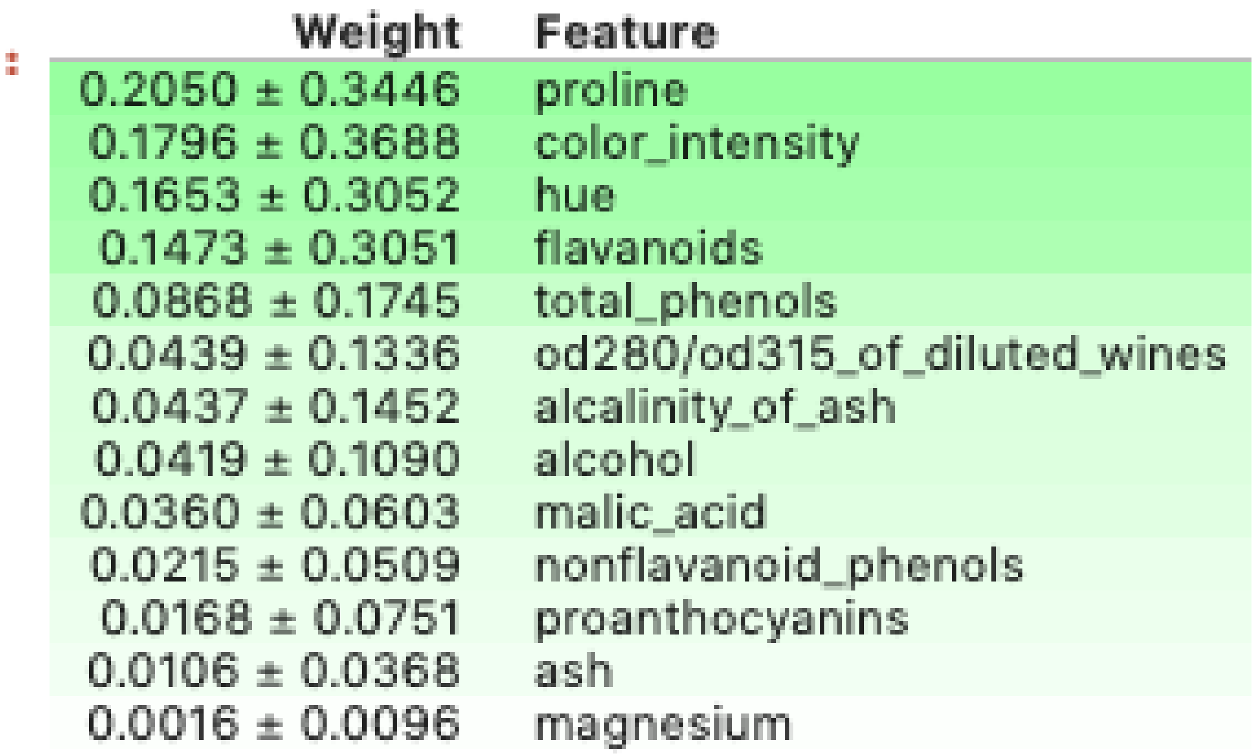


ELI5

ELI5 is another visualisation library that is useful for debugging machine learning models and explaining the predictions they have produced. It works with the most common python machine learning libraries including scikit-learn, XGBoost and Keras.

Let's use ELI5 to inspect the feature importance for the model we trained above.

```
import eli5
eli5.show_weights(model, feature_names = X.columns.tolist())
```

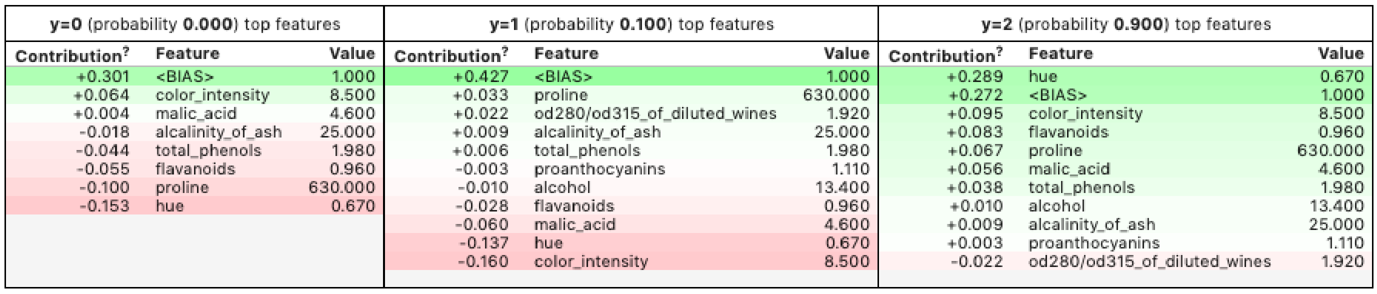


Weight	Feature
0.2050 ± 0.3446	proline
0.1796 ± 0.3688	color_intensity
0.1653 ± 0.3052	hue
0.1473 ± 0.3051	flavanoids
0.0868 ± 0.1745	total_phenols
0.0439 ± 0.1336	od280/od315_of_diluted_wines
0.0437 ± 0.1452	alcalinity_of_ash
0.0419 ± 0.1090	alcohol
0.0360 ± 0.0603	malic_acid
0.0215 ± 0.0509	nonflavanoid_phenols
0.0168 ± 0.0751	proanthocyanins
0.0106 ± 0.0368	ash
0.0016 ± 0.0096	magnesium

By default the `show_weights` method uses `gain` to calculate the weight but you can specify other types by adding the `importance_type` argument.

You can also use `show_prediction` to inspect the reasons for individual predictions.

```
from eli5 import show_prediction
show_prediction(model, X_train.iloc[1], feature_names = X.columns.tolist(),
                show_feature_values=True)
```



y=0 (probability **0.000**) top features

Contribution?	Feature	Value
+0.301	<BIAS>	1.000
+0.064	color_intensity	8.500
+0.004	malic_acid	4.600
-0.018	alcalinity_of_ash	25.000
-0.044	total_phenols	1.980
-0.055	flavanoids	0.960
-0.100	proline	630.000
-0.153	hue	0.670

y=1 (probability **0.100**) top features

Contribution?	Feature	Value
+0.427	<BIAS>	1.000
+0.033	proline	630.000
+0.022	od280/od315_of_diluted_wines	1.920
+0.009	alcalinity_of_ash	25.000
+0.006	total_phenols	1.980
-0.003	proanthocyanins	1.110
-0.010	alcohol	13.400
-0.028	flavanoids	0.960
-0.060	malic_acid	4.600
-0.137	hue	0.670
-0.160	color_intensity	8.500

y=2 (probability **0.900**) top features

Contribution?	Feature	Value
+0.289	hue	0.670
+0.272	<BIAS>	1.000
+0.095	color_intensity	8.500
+0.083	flavanoids	0.960
+0.067	proline	630.000
+0.056	malic_acid	4.600
+0.038	total_phenols	1.980
+0.010	alcohol	13.400
+0.009	alcalinity_of_ash	25.000
+0.003	proanthocyanins	1.110
-0.022	od280/od315_of_diluted_wines	1.920

[LIME](https://github.com/marcotcr/lime)

LIME (local interpretable model-agnostic explanations) is a package for explaining the predictions made by machine learning algorithms. Lime supports explanations for individual predictions from a wide range of classifiers, and support for scikit-learn is built in.

Let's use Lime to interpret some predictions from the model we trained earlier.

Lime can be installed via pip.

```
pip install lime
```

First, we build the explainer. This takes a training dataset as an array, the names of the features used in the model and the names of the classes in the target variable.

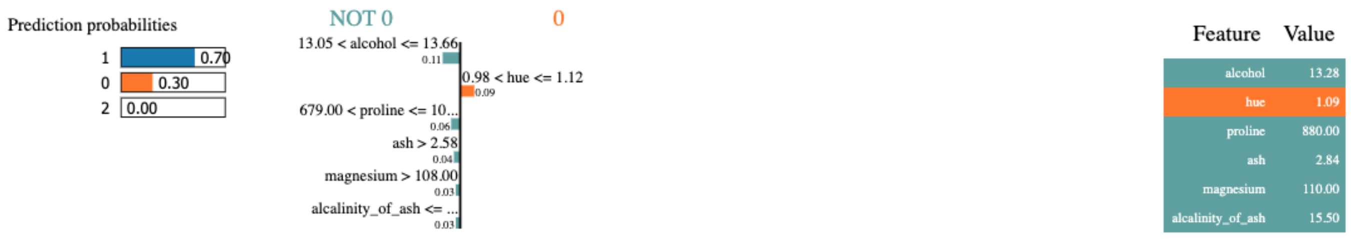
```
import lime.lime_tabular
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
           feature_names=X_train.columns.values.tolist(),
           feature_names=X_train.columns.values.tolist(),
           class_names=y_train.unique())
```

Next, we create a lambda function that uses the model to predict on a sample of the data. This is borrowed from this excellent, more in-depth, tutorial on Lime.

```
predict_fn = lambda x: model.predict_proba(x).astype(float)
```

We then use the explainer to explain the prediction on a selected example. The result is shown below. Lime produces a visualisation showing how the features have contributed to this particular prediction.

```
exp = explainer.explain_instance(X_test.values[0], predict_fn, num_features=6)
exp.show_in_notebook(show_all=False)
```



Feature	Value
alcohol	13.28
hue	1.09
proline	880.00
ash	2.84
magnesium	110.00
alcalinity_of_ash	15.50

MLxtend

This library contains a host of helper functions for machine learning. This covers things like stacking and voting classifiers, model evaluation, feature extraction and engineering and plotting. In addition to the documentation, this paper is a good resource for a more detailed understanding of the package.

Let's use MLxtend to compare the decision boundaries for a voting classifier against its constituent classifiers.

Again it can be installed via pip.

```
pip install mlxtend
```

The imports I am using are shown below.

```
from mlxtend.plotting import plot_decision_regions
from mlxtend.classifier import EnsembleVoteClassifier
import matplotlib.gridspec as gridspec
import itertoolsfrom sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

The following visualisation only works with two features at a time so we will first create an array containing the features `proline` and `color_intensity`. I have chosen these as they had the highest weighting from all the features we inspected earlier using ELI5.

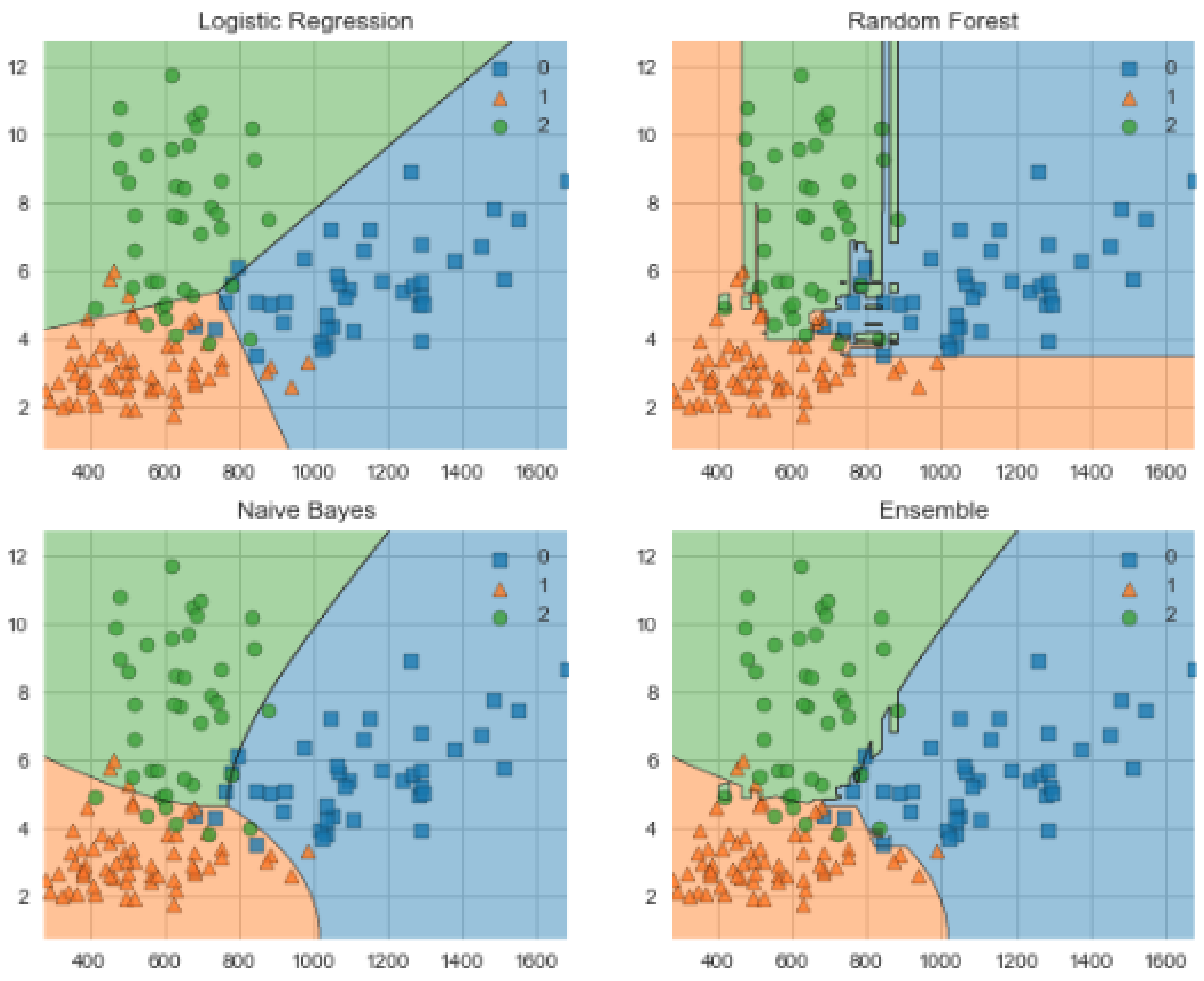
```
X_train_ml = X_train[['proline', 'color_intensity']].values
y_train_ml = y_train.values
```

Next, we create the classifiers, fit them to the training data and visualise the decision boundaries using MLxtend. The output is shown below the code.

```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
```

```
clf3 = GaussianNB()
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2, clf3], weights=[1,1,1])
value=1.5
width=0.75
gs = gridspec.GridSpec(2,2)
fig = plt.figure(figsize=(10,8))
labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'Ensemble']
for clf, lab, grd in zip([clf1, clf2, clf3, eclf],
                         labels,
                         itertools.product([0, 1], repeat=2)):

    clf.fit(X_train_ml, y_train_ml)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X_train_ml, y=y_train_ml, clf=clf)
    plt.title(lab)
```



This is by no means an exhaustive list of libraries for interpreting, visualising and explaining machine learning models. This excellent post contains a long list of other useful libraries to try out.
Thanks for reading!

Bio: Rebecca Vickery is learning data science through self study. Data Scientist @ Holiday Extras. Co-Founder of alGo.

Original. Reposted with permission.

Related:

- Opening Black Boxes: How to leverage Explainable Machine Learning
- A Data Science Playbook for explainable ML/xAI
- Command Line Basics Every Data Scientist Should Know

What do you think?

25 Responses

Upvote Funny Love Surprised

Angry Sad

5 Comments KDnuggets Login

Recommend 1 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Patrick Lee • a year ago

Thanks, interesting article, but when was it written? (It looks to be September 2019, but only because of extra information around the article, I think articles should show their date within themselves, in case external meta data gets changed/lost)?

1 • Reply • Share ›

Gregory Piatetsky Mod Patrick Lee • a year ago

the URL starts with https://www.kdnuggets.com/2...... so this is was written in September 2019.

1 • Reply • Share ›

Patrick Hall • 2 years ago

https://github.com/jphall66...

• Reply • Share ›

jeosol • 2 years ago

There is seems to be a few errors in the syntax and they are due to some of the code lines not breaking
appropriately.

Specifically, for yellowbrick, the second line is not split correctly:

```
import pandas as pd
from sklearn import datasetswine_data = datasets.load_wine()
```

It should be:

```
import pandas as pd
from sklearn import datasets
wine_data = datasets.load_wine()
```

The issue occurs in other code segments.

• Reply • Share ›

<= Previous post
Next post =>

Top Stories Past 30 Days

Most Popular Most Shared

1. **We Don’t Need Data Scientists, We Need Data Engineers**
2. **Are You Still Using Pandas to Process Big Data in 2021? Here are two better options**
3. **Telling a Great Data Story: A Visualization Decision Tree**
4. **Top YouTube Channels for Data Science**
5. **3 Mathematical Laws Data Scientists Need To Know**

1. **We Don’t Need Data Scientists, We Need Data Engineers**
2. **Telling a Great Data Story: A Visualization Decision Tree**
3. **Top YouTube Channels for Data Science**
4. **Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall**
5. **3 Mathematical Laws Data Scientists Need To Know**

Latest News

- Learning from machine learning mistakes
- AI in Dating: Can Algorithms Help You Find Love?
- How to build a DAG Factory on Airflow
- Wrangle Summit 2021: All the Best People, Ideas, and Te...
- More Data Science Cheatsheets
- How to frame the right questions to be answered using data

Top Stories Last Week

Most Popular

1. **How To Overcome The Fear of Math and Learn Math For Data Science**
2. **Know your data much faster with the new Sweetviz Python library**
3. **Must Know for Data Scientists and Data Analysts: Causal Design Patterns**
4. **Are You Still Using Pandas to Process Big Data in 2021? Here are two better options**
5. **3 Mathematical Laws Data Scientists Need To Know**

Most Shared

1. **How To Overcome The Fear of Math and Learn Math For Data Science**
2. **A Machine Learning Model Monitoring Checklist: 7 Things to Track**
3. **Must Know for Data Scientists and Data Analysts: Causal Design Patterns**
4. **Know your data much faster with the new Sweetviz Python library**
5. **4 Machine Learning Concepts I Wish I Knew When I Built My First Model**

More Recent Stories

- How to frame the right questions to be answered using data
- A Simple Way to Time Code in Python
- Data Annotation: tooling & workflows latest trends
- Automating Machine Learning Model Optimization
- Introducing dbt, the ETL and ELT Disrupter
- How to Begin Your NLP Journey
- KDnuggets 21:n11, Mar 17: Is Data Scientist still a satisfy...
- Natural Language Processing Pipelines, Explained
- Metric Matters, Part 1: Evaluating Classification Models
- Data Validation and Data Verification – From Dictionary ...
- Sudoku Rules: Using A Decision Engine To Solve Sudoku
- Are you satisfied in your job? Take our Data Community Job Sat...
- 10 Amazing Machine Learning Projects of 2020
- Forget Telling Stories; Help People Navigate
- Top Stories, Mar 8-14: How To Overcome The Fear of Math and Le...
- AI Industry Innovation: Making the Invisible Visible
- Kedro-Airflow: Orchestrating Kedro Pipelines with Airflow
- Must Know for Data Scientists and Data Analysts: Causal Design...
- Know your data much faster with the new Sweetviz Python library **[Silver Blog]**
- Top February Stories: We Don’t Need Data Scientists, We ...

KDnuggets Home » News » 2019 » Sep » Tutorials, Overviews » Python Libraries for Interpretable Machine Learning (19:n34)

Subscribe to KDnuggets News
X