packages

```
from sklearn.metrics import auc, roc_curve, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams.update({'figure.autolayout': True})
%matplotlib inline
```

fit

```
# use get_dummies to convert categories into columns one hot encode
data_dummy = pd.get_dummies(data, drop_first=True)

#split into train and test to avoid overfitting
np.random.seed(4684)
train, test = train_test_split(data_dummy, test_size = 0.34)
# test size is a fraction of the test sample

# set random forester classifier
# build the model
rf = RandomForestClassifier(n_estimators=100, max_features=3, class_weight={0:xx, 1:yy}, oob_scc
rf.fit(train.drop('label column', axis=1), train['label column'])
```

out-of-bag confusion matrix

rf.oob_decision_function_ (an array that has shape nsample * nlabel_category)

```
# out of bag accuracy
# if 0 and 1 label, rf.oob_decision_function_[:,1].round() can be used to have a 0.5 threshold
print(
"OOB accuracy is",
rf.oob_score_, "\n",
"OOB Confusion Matrix", "\n",
pd.DataFrame(confusion_matrix(train['label column'], rf.oob_decision_function_[:,1].round(), lat
)
# .round() is for 0.5 threshold
```

test confusion matrix

```
print(
"Test accuracy is", rf.score(test.drop('label col', axis=1),test['label col']), "\n",
"Test Set Confusion Matrix", "\n",
pd.DataFrame(confusion_matrix(test['label col'], rf.predict(test.drop('label col', axis=1)), lat
)
```

feature importance plot

```
feat_importances = pd.Series(rf.feature_importances_, index=train.drop('label col', axis=1).colu
feat_importances.sort_values().plot(kind='barh')
```

partial dependence plot

category variable

```
from pdpbox import pdp, info_plots
# category feature: cat0, cat1, cat2....
# cat0 is dropped already if drop_first = true
pdp_iso = pdp.pdp_isolate(model=rf,
                          dataset=train.drop(['label col'], axis=1),
                          model_features=list(train.drop(['label col'], axis=1)),
                          feature=['cat1', 'cat2', 'cat3'....],
                          num_grid_points=50)
pdp_dataset = pd.Series(pdp_iso.pdp, index=pdp_iso.display_columns)
pdp_dataset.sort_values(ascending=False).plot(kind='bar', title='category feature xxx')
plt.show()
```

continuous variable

```
pdp_iso = pdp.pdp_isolate( model=rf,
                          dataset=train.drop(['label col and unnecessary cols'], axis=1),
                          model_features=list(train.drop(['label col and unnecessary cols'], axi
                          feature='continuous feature',
                          num_grid_points=50)
pdp_dataset = pandas.Series(pdp_iso.pdp, index=pdp_iso.feature_grids)
pdp_dataset.plot(title='continuous feature')
plt.show()
```

roc-auc

```
train_fpr, train_tpr, train_thres = roc_curve(train['label'], rf.oob_decision_function_[:,1])
test_fpr, test_tpr, test_thres = roc_curve(test['label'], rf.predict_proba(test.drop('laebl', ax
train_auc = np.round(auc(train_fpr, train_tpr), 3)
test_auc = np.round(auc(test_fpr, test_tpr), 3)

fig, ax = plt.subplots(figsize=(10,8))
ax.plot(train_fpr, train_tpr, label='Train AUC: ' + str(train_auc))
ax.plot(test_fpr, test_tpr, label='Test AUC: ' + str(test_auc))
ax.plot([0,1], [0,1], 'k--')
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.legend(fontsize=12)
plt.show()
```

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
#Let's get false positive rate and true positive rate and plot them in the ROC
fpr, tpr, thresholds = roc_curve(test['class'],pred_prob)
#For consistency with R, we will focus on class errors, defined as class0_error = fpr and class1
error_cutoff=pandas.DataFrame({'cutoff':pandas.Series(thresholds),
                               'class0_error':pandas.Series(fpr),
                               'class1_error': 1 - pandas.Series(tpr)
                                })
error_cutoff['optimal_value'] = 1 - error_cutoff['class1_error'] - error_cutoff['class0_error']
print(error_cutoff.sort_values('optimal_value', ascending=False).head(1))
```

precision_recall_curve

```
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)

def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold, add the legend, axis label and grid
    plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
    plt.show()
```

f1, presion, recall

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

```
from sklearn.metrics import f1_score
f1_score(y_train_5, y_train_pred)
```

maximize f1

```
f1_scores = 2*recall*precision/(recall+precision)
print('Best threshold: ', thresholds[np.argmax(f1_scores)])
print('Best F1-Score: ', np.max(f1_scores))
```