slides_classification

June 18, 2019

1 Playground for Classification Slides

- Stephen W. Thomas
- Used for MMA 869, MMAI 869, and GMMA 869

In [1]:
```
import datetime
print(datetime.datetime.now())
```

```
2019-04-24 16:21:30.024552
```

In [2]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

from sklearn.metrics import silhouette_score, silhouette_samples
import sklearn.metrics
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture

import itertools

import scipy

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# This will ensure that matplotlib figures don't get cut off when saving with savefig()
#from matplotlib import rcParams
#rcParams.update({'figure.autolayout': True})
```

```
C:\Users\st50\AppData\Roaming\Python\Python36\site-packages\matplotlib\__init__.py:886: Matplot
examples.directory is deprecated; in the future, examples will be found relative to the 'datapa
  "found relative to the 'datapath' directory.".format(key))
```

2 Other Datasets to Consider, but not used yet

In [3]:
```
# Bank Marketing Data
#df = pd.read_csv("https://raw.githubusercontent.com/stepthom/sandbox/master/data/bank
#df = df.rename(index=str, columns={"y": "bought"})

# Default dataset from ISLR
#df = pd.read_csv("https://raw.githubusercontent.com/stepthom/sandbox/master/data/islr
#df = df.drop(["Unnamed: 0"], axis=1)
```

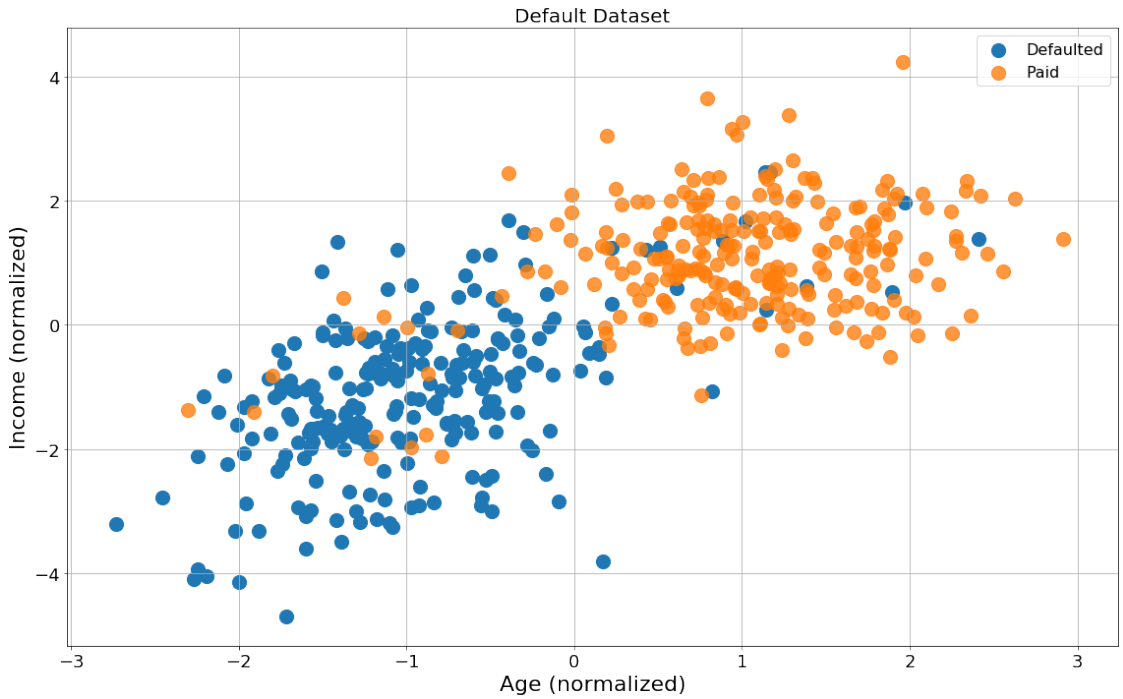
3 Generate Data

For educational purposes, we'll generate a synthetic dataset, rather than using a real one, at first. We'll create a dataset that has two features, both informative, with some overlap, but not much.

Later, we'll graduate to using real-world datasets with more features and less seperation between classes.

In [4]:
```
from sklearn.datasets import make_classification
import random

# After experiementation, this random state generates a "good looking" dataset
r = 4184

X, y = make_classification(n_samples=500, n_features=2, n_redundant=0, n_informative=2
                           n_clusters_per_class=1, flip_y=0.09, class_sep = 1.1, rando

X1 = pd.DataFrame(X, columns=['Age', 'Income'])
y1 = pd.Series(y, name='Default')
df = pd.concat([X1, y1], axis=1)
```

In [5]:
```
plt.figure(figsize=(16, 10));
plt.grid(True);

ind_d = y==1
ind_p = y==0
plt.scatter(X[ind_d,0], X[ind_d,1], marker='o', s=200, label='Defaulted');
plt.scatter(X[ind_p,0], X[ind_p,1], marker='o', s=200, label="Paid", alpha=0.8);

plt.legend(fontsize=16);
plt.title("Default Dataset", fontsize=20);
plt.xlabel('Age (normalized)', fontsize=22);
plt.ylabel('Income (normalized)', fontsize=22);
plt.xticks(fontsize=18);
plt.yticks(fontsize=18);
plt.tight_layout();
plt.savefig('out/default-data.png');
```




```
In [6]: X.shape
        X[1:10,:]
        y.shape
        y[1:10]
        df.head(15)

Out[6]: (500, 2)

Out[6]: array([[-1.64575979, -2.9333321 ],
               [ 0.27982946,  0.83540131],
               [ 0.19585293,  1.24920912],
               [-1.35705338, -1.4091226 ],
               [-0.49425018, -3.00562239],
               [-1.37872953,  0.4408266 ],
               [ 1.14567439,  0.24986299],
               [ 0.96821192,  0.57301515],
               [ 1.56205843, -0.04601173]])

Out[6]: (500,)

Out[6]: array([1, 0, 0, 1, 1, 0, 1, 0, 0])

Out[6]:          Age    Income  Default
        0   0.748126  1.185890        0
        1  -1.645760 -2.933332        1
```

```
2   0.279829  0.835401        0
3   0.195853  1.249209        0
4  -1.357053 -1.409123        1
5  -0.494250 -3.005622        1
6  -1.378730  0.440827        0
7   1.145674  0.249863        1
8   0.968212  0.573015        0
9   1.562058 -0.046012        0
10  2.336449  2.159094        0
11 -0.601315  1.114726        1
12 -1.739327 -2.246285        1
13  0.955808  0.585645        0
14 -0.668198 -0.615540        1
```

3.1 Profile the Data

```
In [7]: import pandas_profiling

        pandas_profiling.ProfileReport(df, check_correlation=False)
```

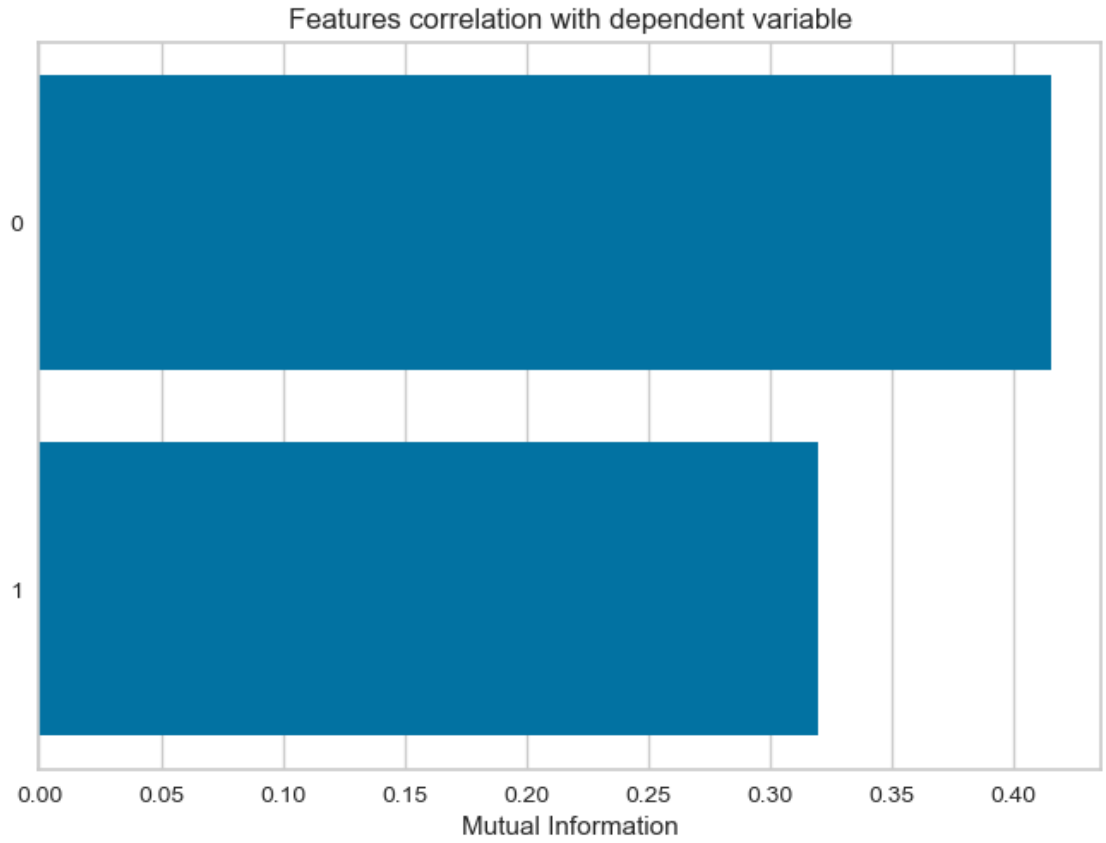
```
C:\Users\st50\AppData\Local\Continuum\anaconda3\envs\small_sklearn\lib\site-packages\pandas_pro
  matplotlib.use(BACKEND)
```

```
Out[7]: <pandas_profiling.ProfileReport at 0x246914c1a90>
```

```
In [8]: from yellowbrick.target import FeatureCorrelation

        visualizer = FeatureCorrelation(method='mutual_info-classification', sort=True)
        visualizer.fit(X, y, random_state=0)
        visualizer.poof()
```

```
C:\Users\st50\AppData\Roaming\Python\Python36\site-packages\matplotlib\__init__.py:886: Matplot
examples.directory is deprecated; in the future, examples will be found relative to the 'datapa
  "found relative to the 'datapath' directory.".format(key))
```

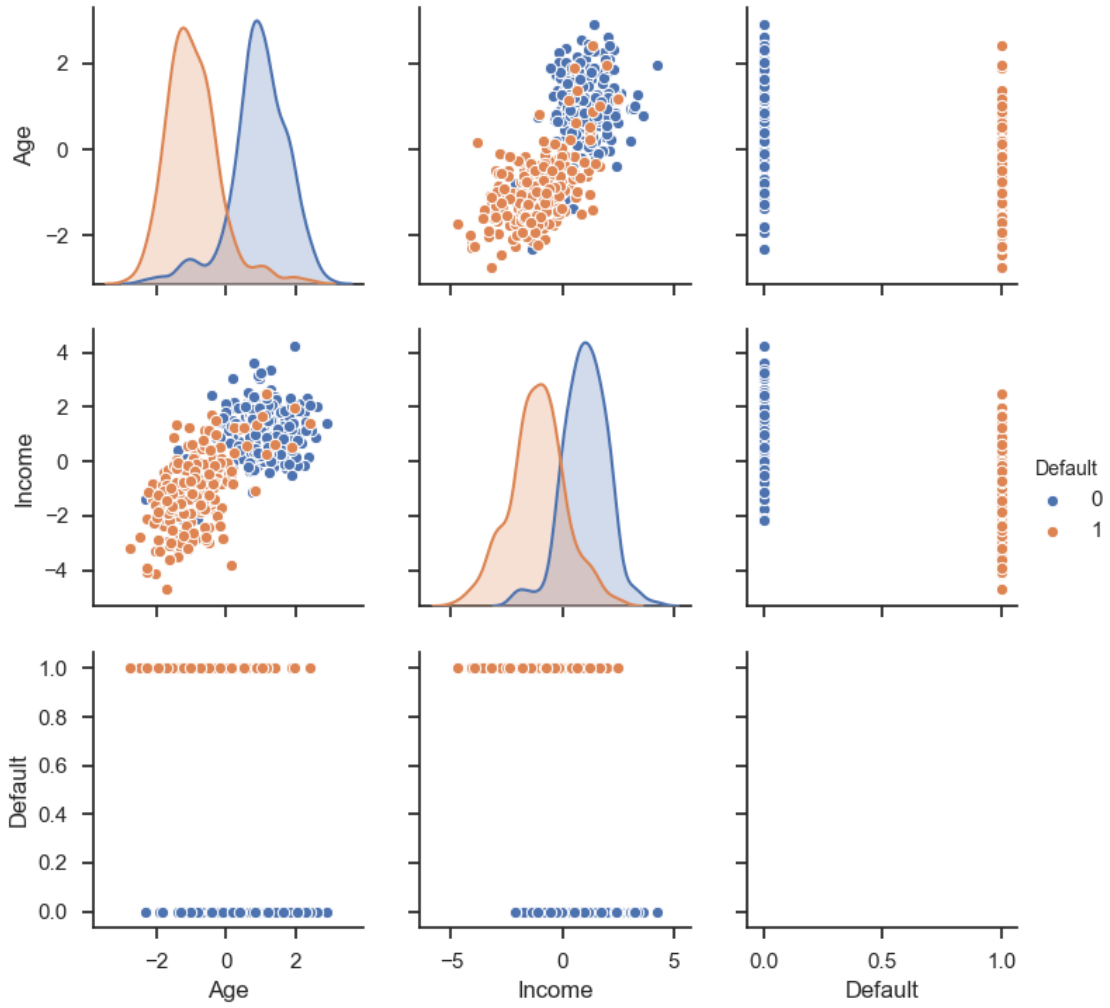
```
Out[8]: FeatureCorrelation(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x0000024691527
                   feature_index=None, feature_names=None, labels=None,
                   method='mutual_info-classification', sort=True)
```



3.2 Plot the Data

```
In [9]: import seaborn as sns
        sns.set(style="ticks")

        sns.pairplot(df, hue="Default");
```

```
C:\Users\st50\AppData\Local\Continuum\anaconda3\envs\small_sklearn\lib\site-packages\statsmodel
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\Users\st50\AppData\Local\Continuum\anaconda3\envs\small_sklearn\lib\site-packages\statsmodel
  FAC1 = 2*(np.pi*bw/RANGE)**2
```



4 Splitting the Data

```
In [10]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

5 Helper Functions for Plotting Decision Boundary, etc.

```
In [11]: from matplotlib.colors import ListedColormap
         from sklearn.metrics import roc_curve, auc

         # Adopted from: https://scikit-learn.org/stable/auto_examples/classification/plot_cla

         def plot_boundaries(X_train, X_test, y_train, y_test, clf, clf_name, ax, hide_ticks=Tr
```

```
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])

    X = np.concatenate((X_train, X_test), axis=0)
    y = np.concatenate((y_train, y_test), axis=0)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))


    score = clf.score(X_test, y_test);

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]);
    else:
        Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1];

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8);

    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=100, cmap=cm_bright, edgeco
    # Plot the testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=50, cmap=cm_bright, edgecolors

    ax.set_xlim(xx.min(), xx.max());
    ax.set_ylim(yy.min(), yy.max());
    if hide_ticks:
        ax.set_xticks(());
        ax.set_yticks(());
    else:
        ax.tick_params(axis='both', which='major', labelsize=18)
        #ax.yticks(fontsize=18);

    ax.set_title(clf_name, fontsize=28);
    ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'), size=35, horiz
    ax.grid();



def plot_roc(clf, X_test, y_test, name, ax, show_thresholds=True):
    y_pred_rf = clf.predict_proba(X_test)[:, 1]
    fpr, tpr, thr = roc_curve(y_test, y_pred_rf)
```

```
    ax.plot([0, 1], [0, 1], 'k--');
    ax.plot(fpr, tpr, label='{}, AUC={:.2f}'.format(name, auc(fpr, tpr)));
    ax.scatter(fpr, tpr);

    if show_thresholds:
        for i, th in enumerate(thr):
            ax.text(x=fpr[i], y=tpr[i], s="{:.2f}".format(th), fontsize=14,
                    horizontalalignment='left', verticalalignment='top', color='blac
                    bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,p

    ax.set_xlabel('False positive rate', fontsize=18);
    ax.set_ylabel('True positive rate', fontsize=18);
    ax.tick_params(axis='both', which='major', labelsize=18);
    ax.grid(True);
    ax.set_title('ROC Curve', fontsize=18)
```

6 Decision Trees

```
In [12]: from sklearn.tree import DecisionTreeClassifier

         clf = DecisionTreeClassifier(random_state=42, criterion="entropy",
                                      min_samples_split=10, min_samples_leaf=10, max_depth=3, m
         clf.fit(X_train, y_train)

         y_pred_dt = clf.predict(X_test)
```

```
Out[12]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
                     max_features=None, max_leaf_nodes=5, min_impurity_decrease=0.0,
                     min_impurity_split=None, min_samples_leaf=10,
                     min_samples_split=10, min_weight_fraction_leaf=0.0,
                     presort=False, random_state=42, splitter='best')
```

```
In [13]: feature_names = X1.columns
         class_names = [str(x) for x in clf.classes_]
```

6.1 Use the Model to Predict Someone New

```
In [14]: clf.predict_proba([[2, 2]])
         clf.predict([[2, 2]])
```

```
Out[14]: array([[0.93922652, 0.06077348]])
```

```
Out[14]: array([0])
```

6.2 Model Parameters

Surpisingly, sci-kit learn does not have a function to print the decision tree in text format. (It does have a way to graphical render the three, which we'll do later.) For now, we'll just print a few stats about the tree.

```
In [15]: print(clf.tree_.node_count)
         print(clf.tree_.impurity)
         print(clf.tree_.children_left)
         print(clf.tree_.threshold)
```

```
9
[0.99971144 0.40077522 0.45449306 0.99107606 0.33050773 0.22746906
 0.83147439 0.28998605 0.        ]
[ 1  5  3 -1 -1  7 -1 -1 -1]
[-0.05775159 -0.15540861  0.2254563  -2.         -2.         -0.7764757
 -2.         -2.         -2.        ]
```

6.3 Model Performance

```
In [16]: from pandas_ml import ConfusionMatrix

         print(ConfusionMatrix(y_test, y_pred_dt))
```

```
Predicted  False  True  __all__
Actual
False         50     5       55
True           5    40       45
__all__       55    45      100
```

```
In [17]: from sklearn.metrics import classification_report

         print(classification_report(y_test, y_pred_dt, target_names=class_names))
```

```
             precision    recall  f1-score   support

          0       0.91      0.91      0.91        55
          1       0.89      0.89      0.89        45

avg / total       0.90      0.90      0.90       100
```

```
In [18]: from sklearn.metrics import accuracy_score, cohen_kappa_score, f1_score, log_loss

         print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_dt)))
         print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_dt)))
         print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_dt)))
         print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_dt)))
```

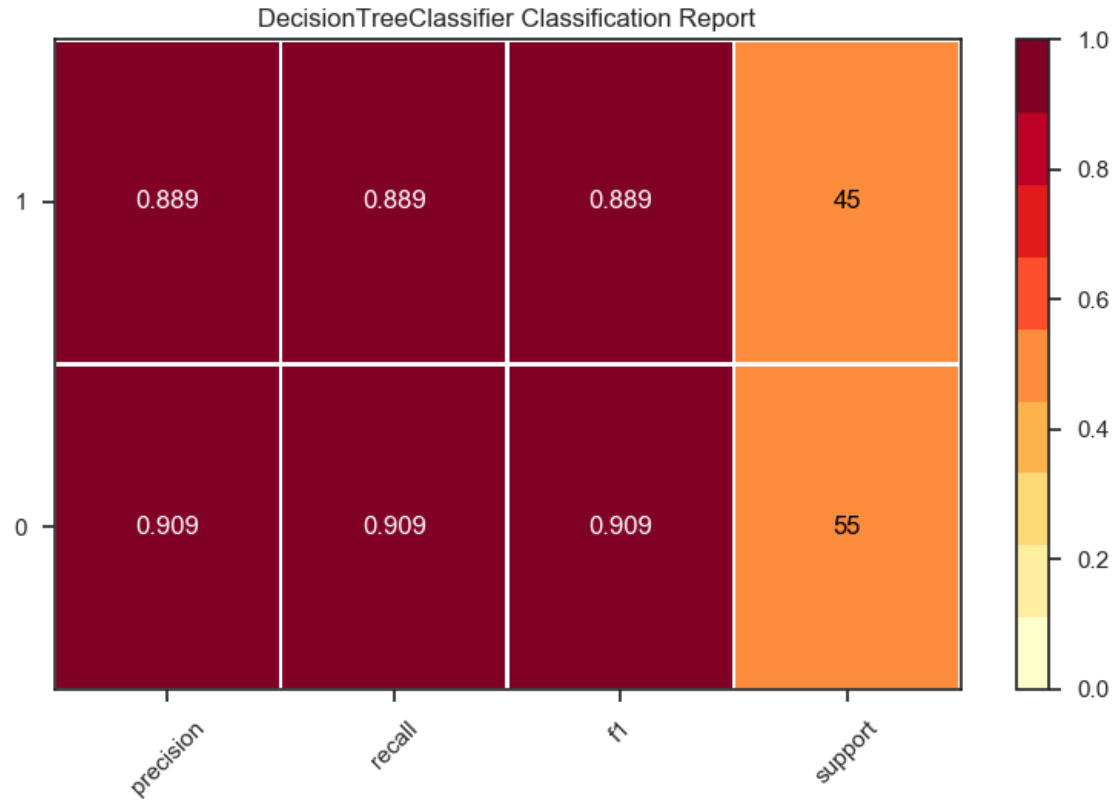
```
Accuracy = 0.90
Kappa = 0.80
```

```
F1 Score = 0.89
Log Loss = 3.45
```

```
In [19]: from yellowbrick.classifier import ClassificationReport

         # Instantiate the classification model and visualizer
         visualizer = ClassificationReport(clf, classes=class_names, support=True)

         visualizer.fit(X_train, y_train)  # Fit the visualizer and the model
         visualizer.score(X_test, y_test)  # Evaluate the model on the test data
         g = visualizer.poof()             # Draw/show/poof the data
```

```
Out[19]: ClassificationReport(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x0000024692
                   classes=None,
                   cmap=<matplotlib.colors.ListedColormap object at 0x00000246926A6F60>,
                   model=None, support=True)
```

```
Out[19]: 0.9
```

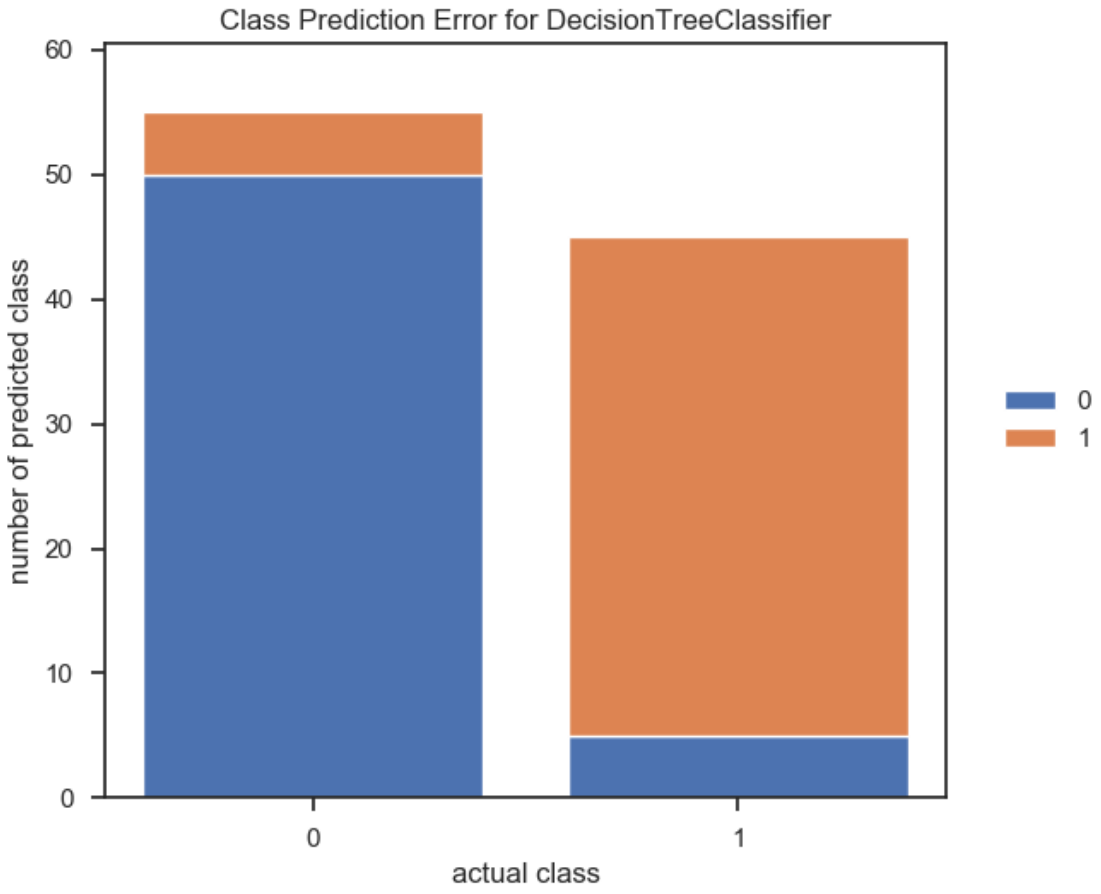



```
In [20]: from yellowbrick.classifier import ClassPredictionError
```

```
visualizer = ClassPredictionError(clf, classes=class_names)

visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
g = visualizer.poof()
```

```
Out[20]: ClassPredictionError(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x0000024692
                  classes=None, model=None)
```

```
Out[20]: 0.9
```



6.3.1 ROC Curve

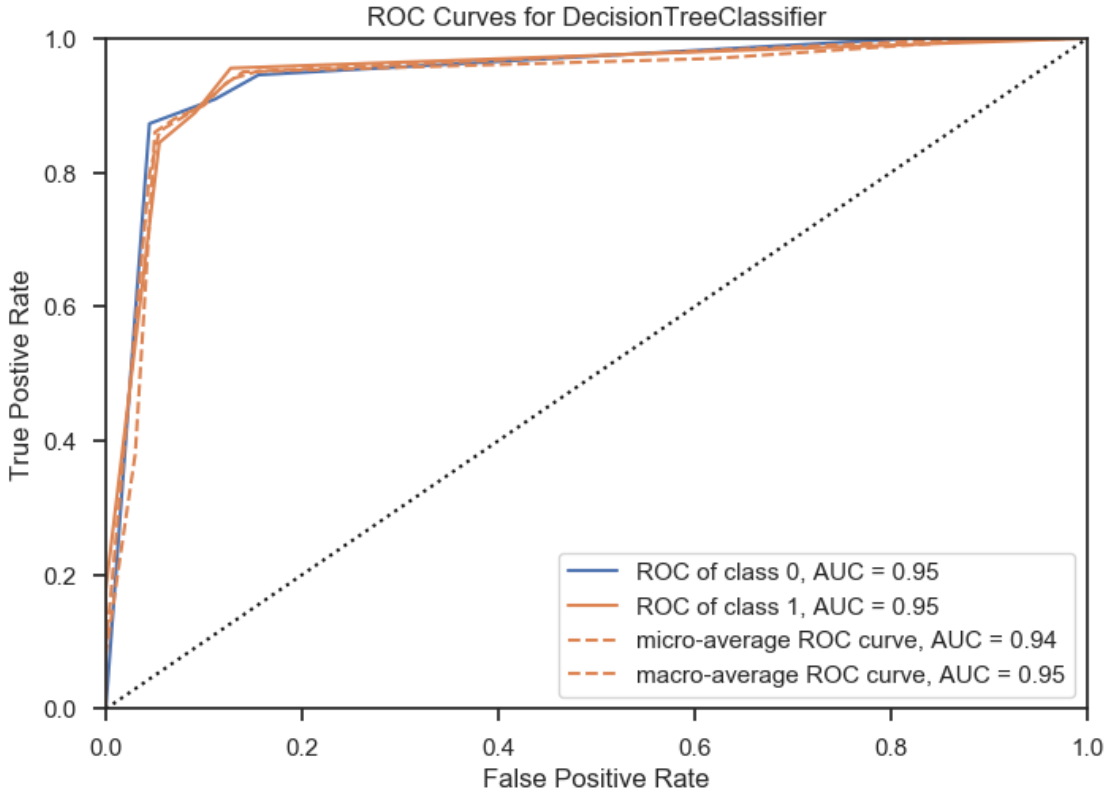
With YellowBrick

```
In [21]: from yellowbrick.classifier import ROCAUC

visualizer = ROCAUC(clf, classes=class_names)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualizer
```

```
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()             # Draw/show/poof the data
```

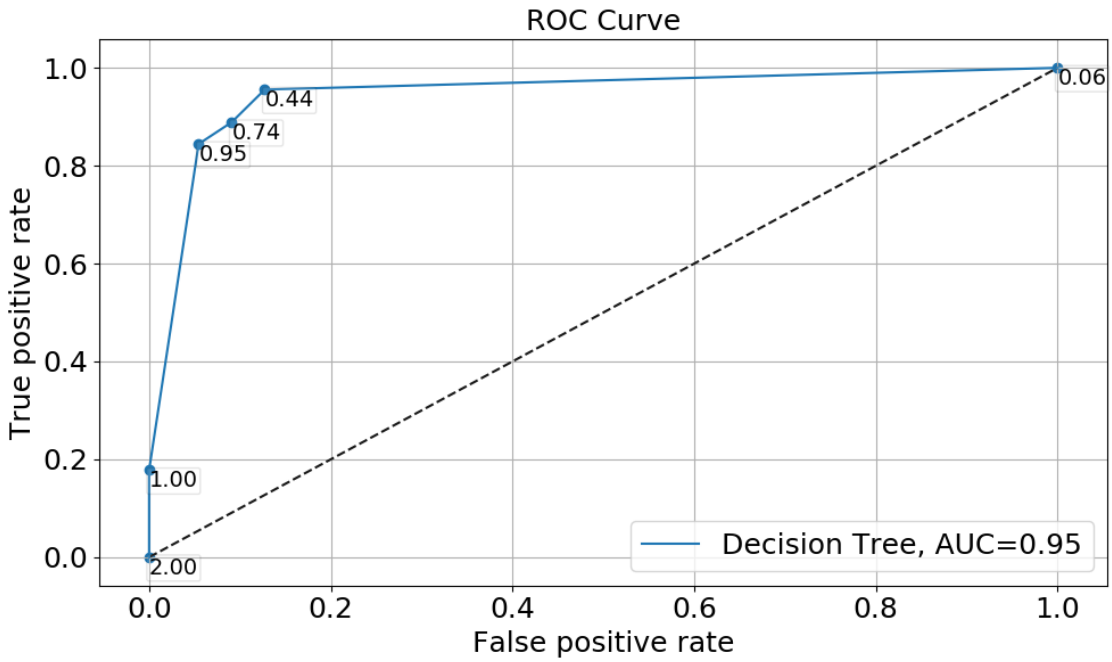
```
Out[21]: ROCAUC(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x00000246927315F8>,
             classes=None, macro=True, micro=True, model=None, per_class=True)
```

```
Out[21]: 0.9
```



Manual

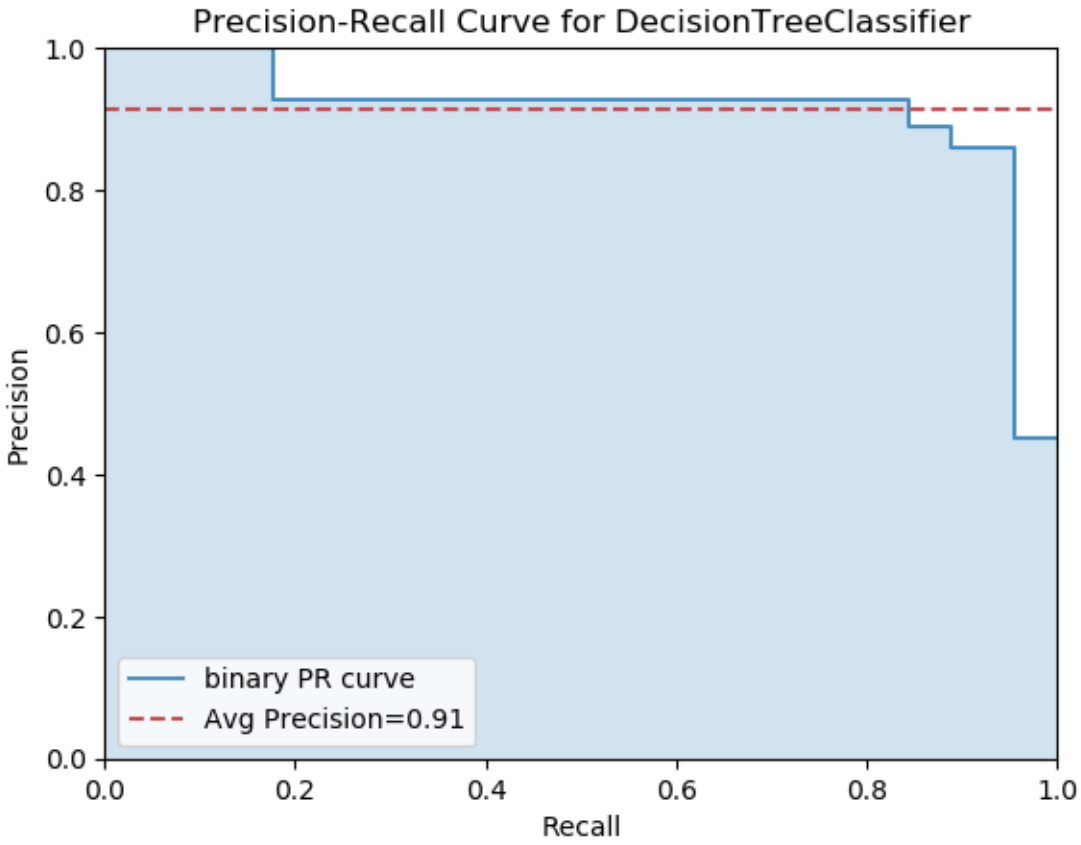
```
In [22]: plt.style.use('default');
         figure = plt.figure(figsize=(10, 6));
         ax = plt.subplot(1, 1, 1);
         plot_roc(clf, X_test, y_test, "Decision Tree", ax)
         plt.legend(loc='lower right', fontsize=18);
         plt.tight_layout();
         plt.savefig('out/default-dt-roc.png');
```



6.3.2 Precision Recall Curve

With YellowBrick

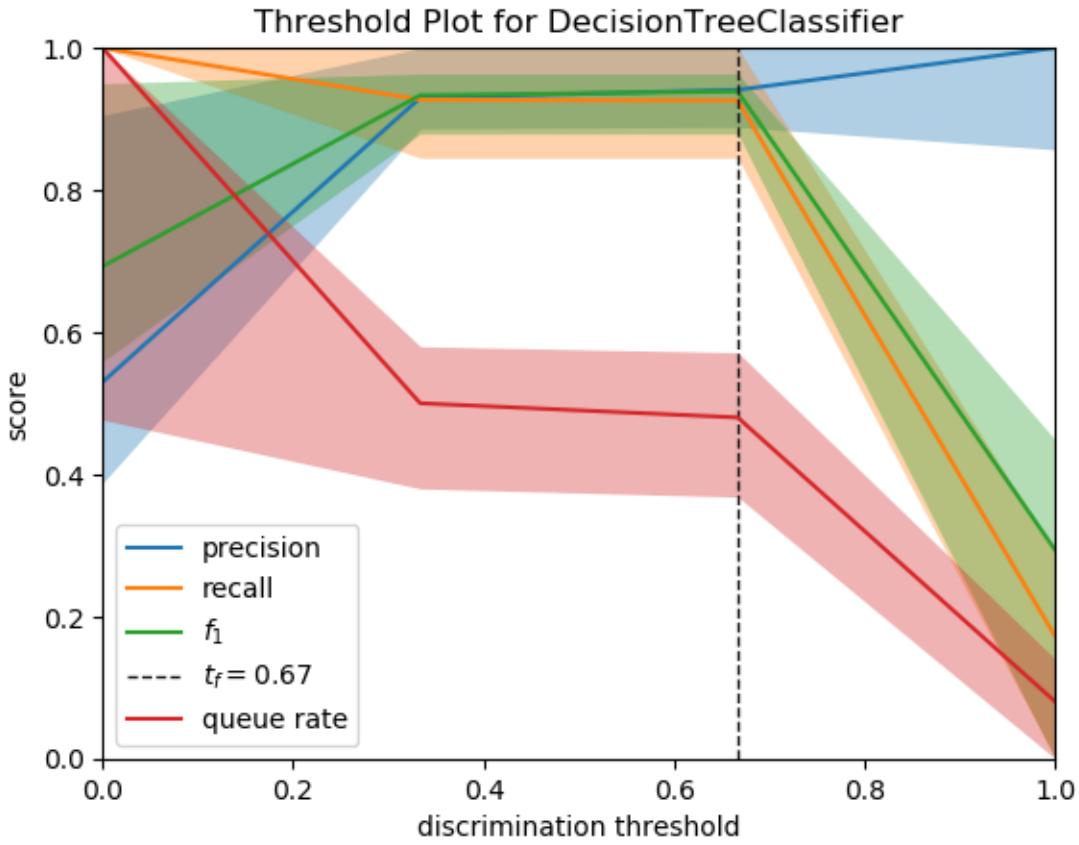
```
In [23]: from yellowbrick.classifier import PrecisionRecallCurve

         viz = PrecisionRecallCurve(clf);
         viz.fit(X_train, y_train);
         viz.score(X_test, y_test);
         viz.poof();
```



6.3.3 Discrimintation Threshold

```
In [24]: from yellowbrick.classifier import DiscriminationThreshold

         visualizer = DiscriminationThreshold(clf)
         visualizer.fit(X, y)  # Fit the training data to the visualizer
         visualizer.poof()     # Draw/show/poof the data
```

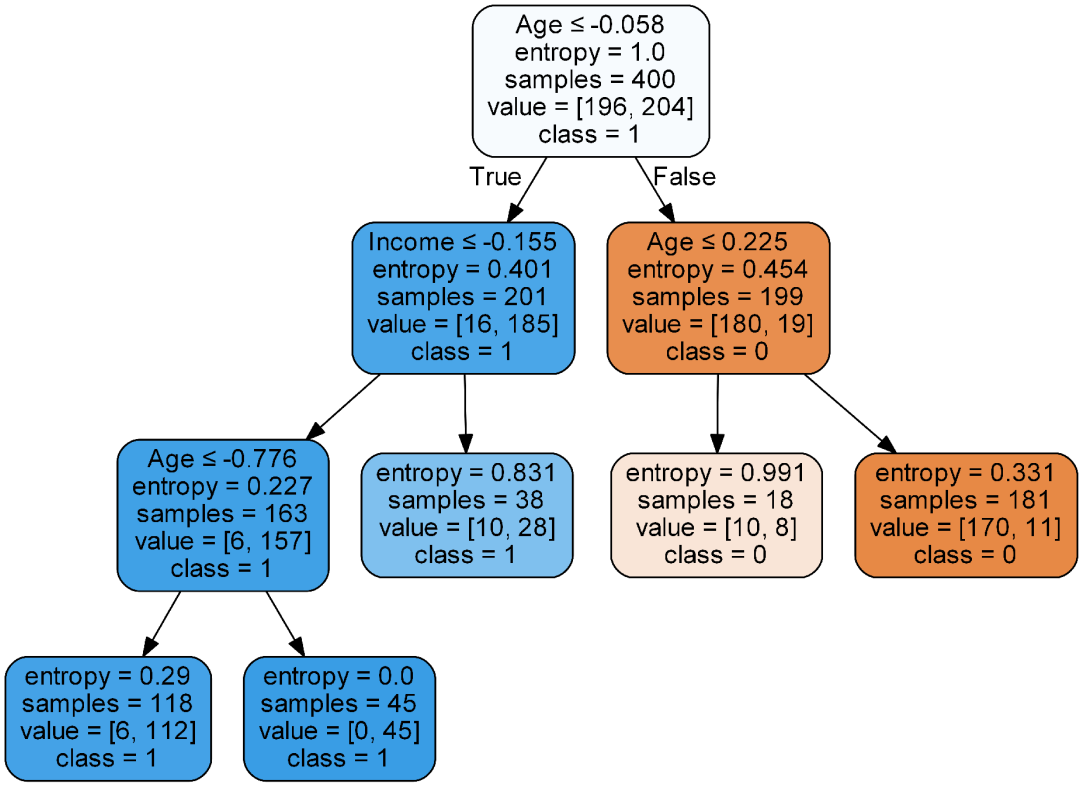
```
Out[24]: DiscriminationThreshold(argmax='fscore',
                     ax=<matplotlib.axes._subplots.AxesSubplot object at 0x0000024692348908>,
                     cv=0.1, exclude=None, fbeta=1.0, model=None, n_trials=50,
                     quantiles=array([0.1, 0.5, 0.9]), random_state=None)
```



6.4 Model Visualization

```
In [25]: from sklearn.tree import export_graphviz
         import graphviz

         base='out/default_dt_graph'
         dot = base + '.dot'
         png = base + '.png'

         dot_data = export_graphviz(clf, out_file=dot, feature_names=feature_names, class_names
                                    filled=True, rounded=True, special_characters=True,
                                    leaves_parallel=False, proportion=False)
         graph = graphviz.Source(dot_data)

         from subprocess import call
         call(['dot', '-Tpng', dot, '-o', png, '-Gdpi=600'])


         # Display in python
         import matplotlib.pyplot as plt
```

```
plt.figure(figsize = (24, 28));
plt.imshow(plt.imread(png));
plt.axis('off');
plt.show();
```



In [26]:
```
import importlib
spec = importlib.util.find_spec("dtreeviz")
dtreeviz_found = spec is not None

if dtreeviz_found:
    from dtreeviz.trees import *

    viz = dtreeviz(clf, X, y, target_name='Default', feature_names=feature_names, cla
    viz.save('out/default_dt_dtreeviz1.svg')
    viz
else:
    print("dtreeviz not installed on this system.")
```

```
dtreeviz not installed on this system.
```

In [27]:
```
if dtreeviz_found:
    viz = dtreeviz(clf, X, y, target_name='Default', feature_names=feature_names, cla
```

```
            viz.save('out/default_dt_dtreeviz2.svg')
            viz
        else:
            print("dtreeviz not installed on this system.")
```

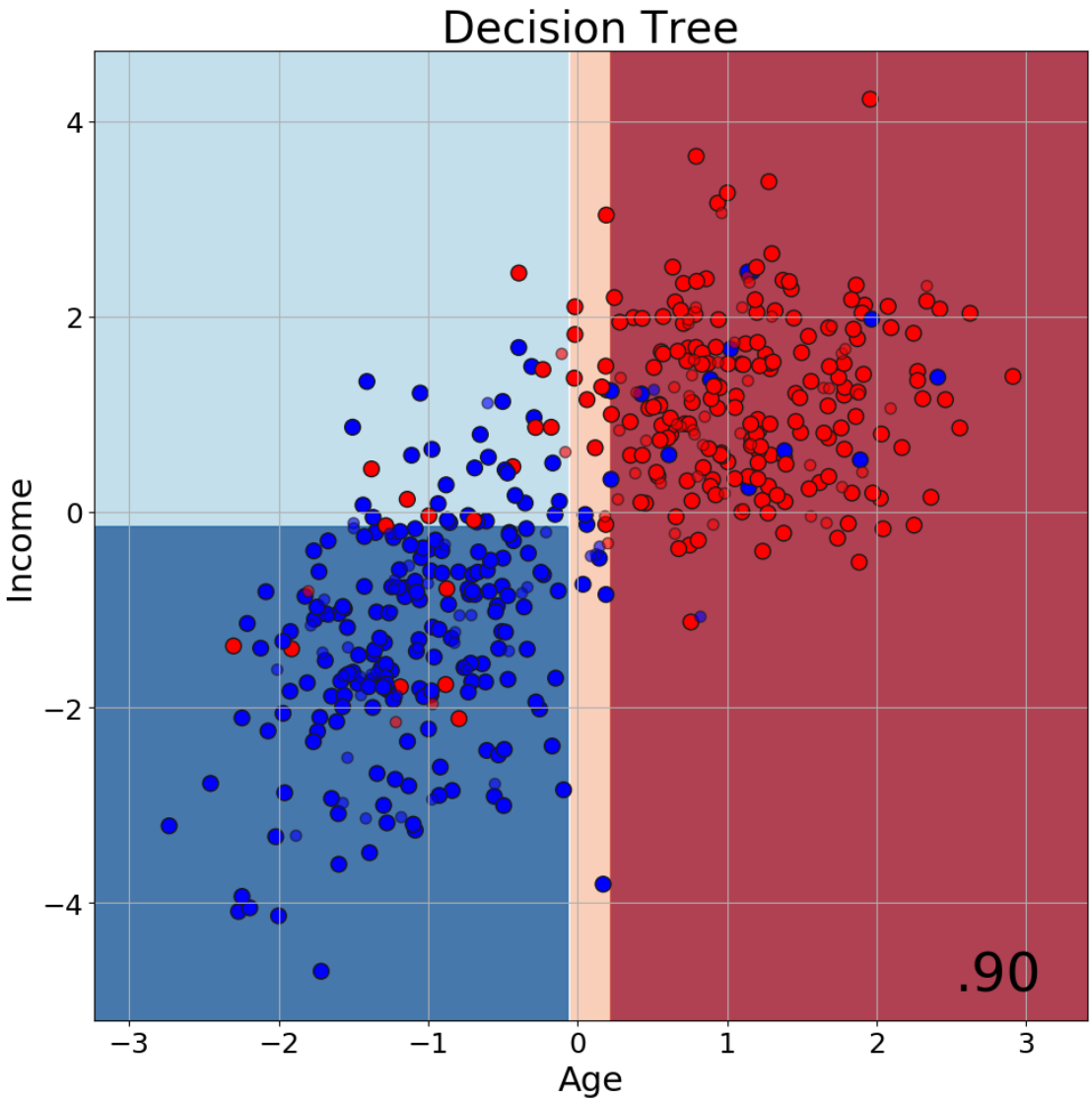
```
dtreeviz not installed on this system.
```

```
In [28]: if dtreeviz_found:
            # Select a random sample
            x = X_train[13,:]
            print(x)
            print(y_train[13])

            viz = dtreeviz(clf, X_train, y_train, target_name='default', feature_names=feature
            viz

        else:
            print("dtreeviz not installed on this system.")
```

```
dtreeviz not installed on this system.
```

6.4.1 Decision Boundary

```
In [29]: figure = plt.figure(figsize=(10, 10));
         ax = plt.subplot(1, 1, 1);
         plot_boundaries(X_train, X_test, y_train, y_test, clf, "Decision Tree", ax, hide_ticks
         ax.set_xlabel("Age", fontsize=22)
         ax.set_ylabel("Income", fontsize=22)
         plt.tight_layout();
         plt.savefig('out/default-dt-5-boundaries.png');
```

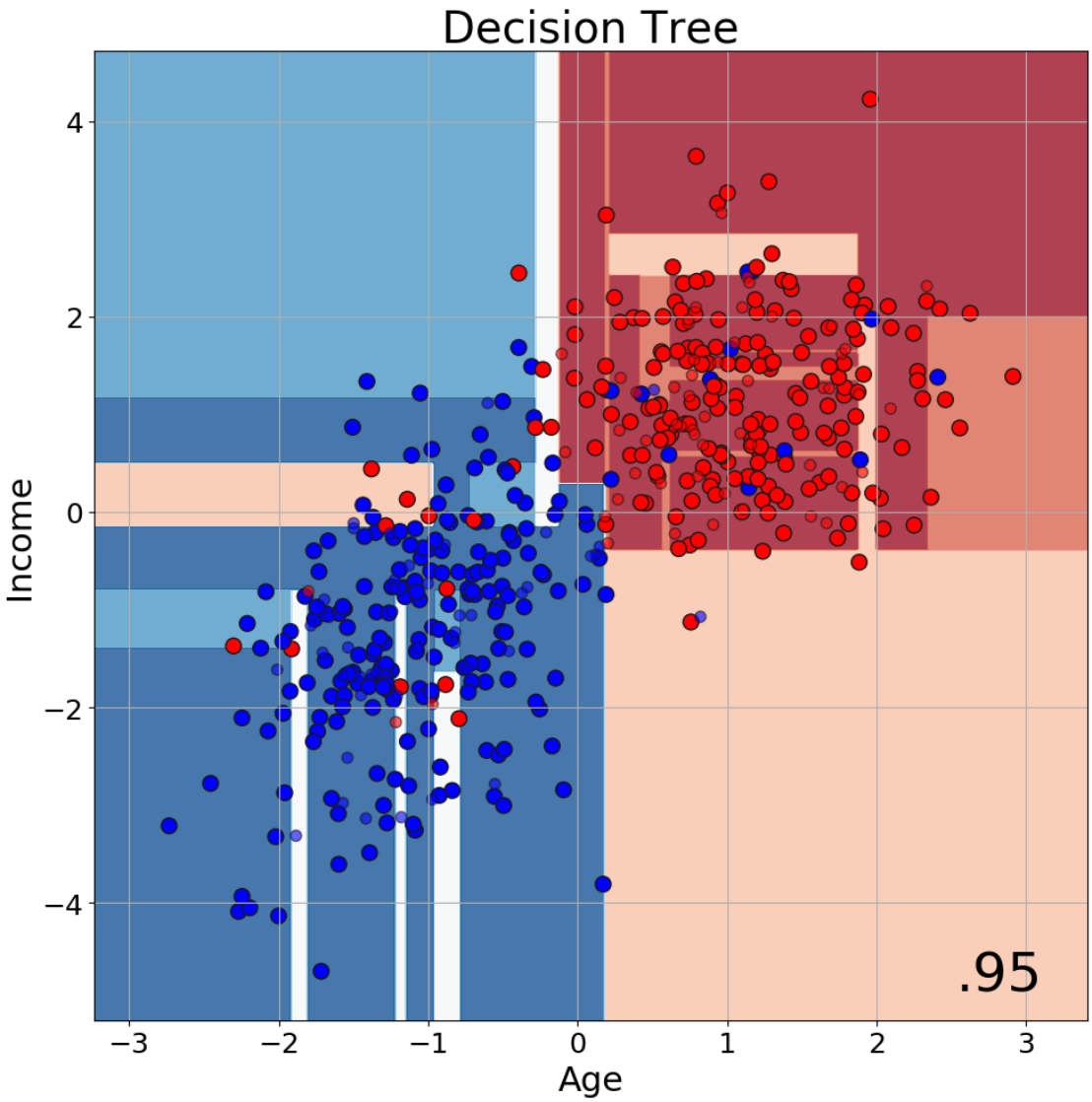


6.5 Model Selection

6.5.1 Grid Search

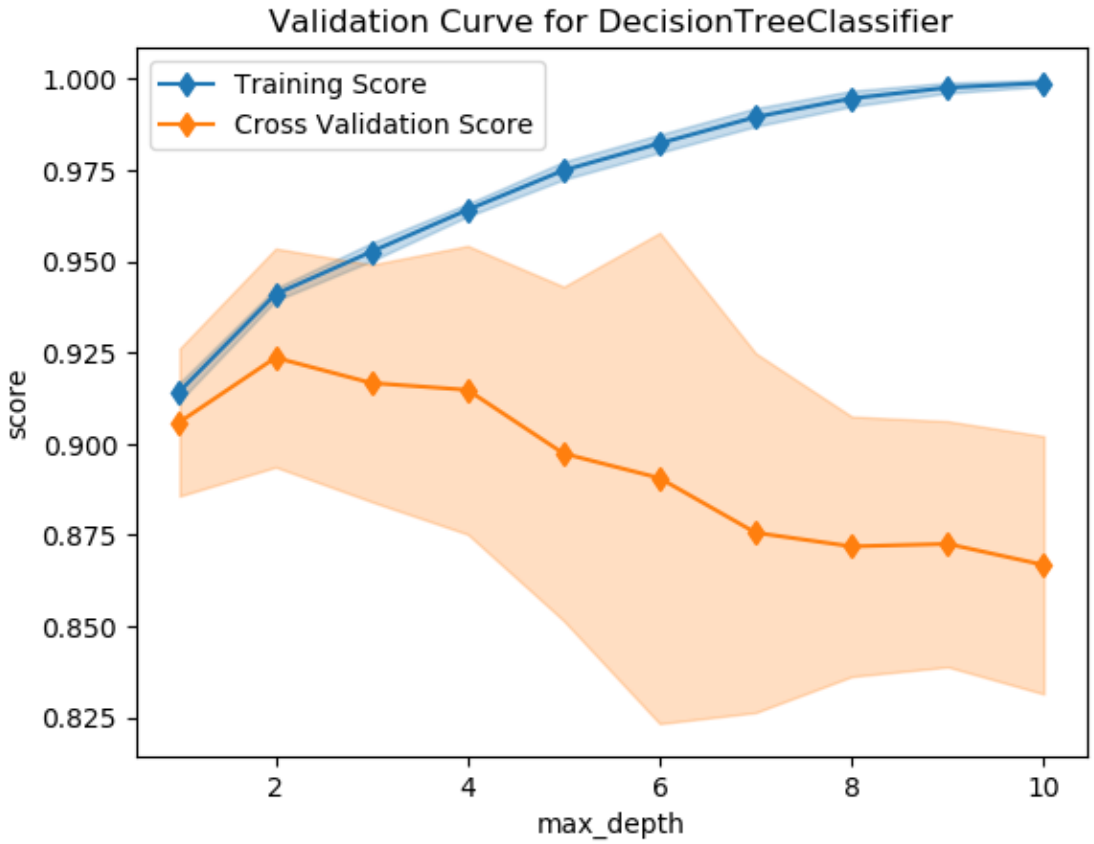
```
In [30]: from sklearn.model_selection import GridSearchCV

         parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}

         treeclf = DecisionTreeClassifier(splitter='best', presort=True, class_weight=None, ra
         parameters = {'criterion':('gini', 'entropy'), 'max_depth':[2, 4, 6, 8, 10], 'min_samp
                       'max_features':[None, 'auto'], 'max_leaf_nodes':[None, 5, 10, 50], 'min_
         cv_clf = GridSearchCV(treeclf, parameters, scoring='roc_auc', cv=5, return_train_score
         %time cv_clf.fit(X, y)
```

```
Wall time: 26.8 s


Out[30]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=True, random_state=42,
                     splitter='best'),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'criterion': ('gini', 'entropy'), 'max_depth': [2, 4, 6, 8, 10], 'm
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring='roc_auc', verbose=0)

In [31]: cv_clf.best_params_
         cv_clf.best_score_
         cv_clf.best_estimator_

Out[31]: {'criterion': 'gini',
          'max_depth': 10,
          'max_features': 'auto',
          'max_leaf_nodes': 50,
          'min_impurity_decrease': 0,
          'min_samples_leaf': 5,
          'min_samples_split': 2}

Out[31]: 0.9392180712284914

Out[31]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                     max_features='auto', max_leaf_nodes=50,
                     min_impurity_decrease=0, min_impurity_split=None,
                     min_samples_leaf=5, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=True, random_state=42,
                     splitter='best')

In [32]: figure = plt.figure(figsize=(10, 10));
         ax = plt.subplot(1, 1, 1);
         plot_boundaries(X_train, X_test, y_train, y_test, cv_clf.best_estimator_, "Decision Tr
         ax.set_xlabel("Age", fontsize=22)
         ax.set_ylabel("Income", fontsize=22)
         plt.tight_layout();
         plt.savefig('out/default-dt-best-boundaries.png');
```




```
In [33]: if dtreeviz_found:
             viz = dtreeviz(cv_clf.best_estimator_, X, y, target_name='Default', feature_names=
             viz

In [34]: from yellowbrick.model_selection import ValidationCurve

         viz = ValidationCurve(DecisionTreeClassifier(), param_name="max_depth", param_range=np
         viz.fit(X, y)
         viz.poof(outpath='out/dt_validation_curve.png')
         viz.poof()

Out[34]: ValidationCurve(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x000002469209E3C8
                 cv=10, groups=None, logx=False, model=None, n_jobs=1,
```

```
param_name='max_depth',
param_range=array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
pre_dispatch='all', scoring='roc_auc')
```



7 Naive Bayes

```
In [35]: from sklearn.naive_bayes import GaussianNB
         gnb = GaussianNB()
         gnb = gnb.fit(X_train, y_train)
         gnb

         y_pred_gnb = gnb.predict(X_test)

Out[35]: GaussianNB(priors=None, var_smoothing=1e-09)

In [36]: from sklearn.metrics import confusion_matrix

         confusion_matrix(y_test, y_pred_gnb)

Out[36]: array([[51,  4],
                [ 2, 43]], dtype=int64)
```

```
In [37]: from sklearn.metrics import classification_report
```

7.1 Model Parameters

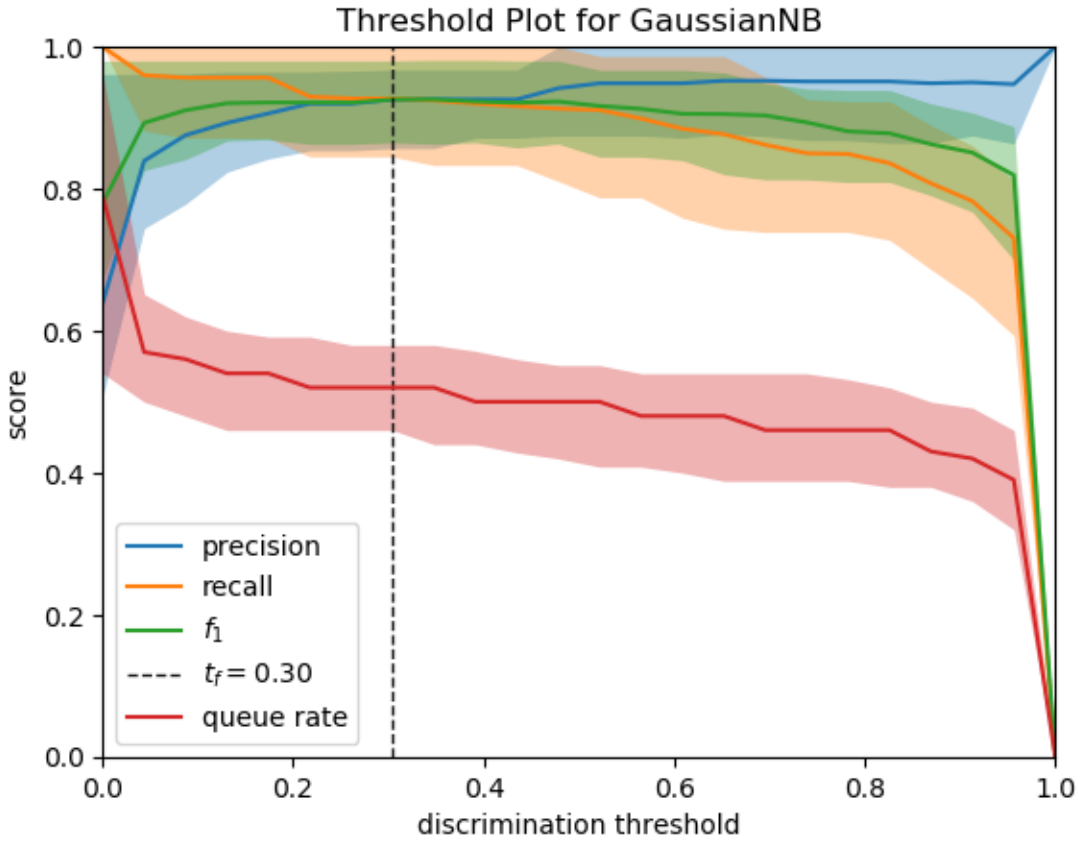
```
In [38]: gnb.theta_ # Mean of each feature per class
         gnb.sigma_ # Variance of each feature per class

Out[38]: array([[ 0.99887777,  1.06512859],
                [-0.91827636, -1.08561939]])

Out[38]: array([[0.70142311, 0.949339  ],
                [0.62737657, 1.68858577]])
```

7.2 Model Performance

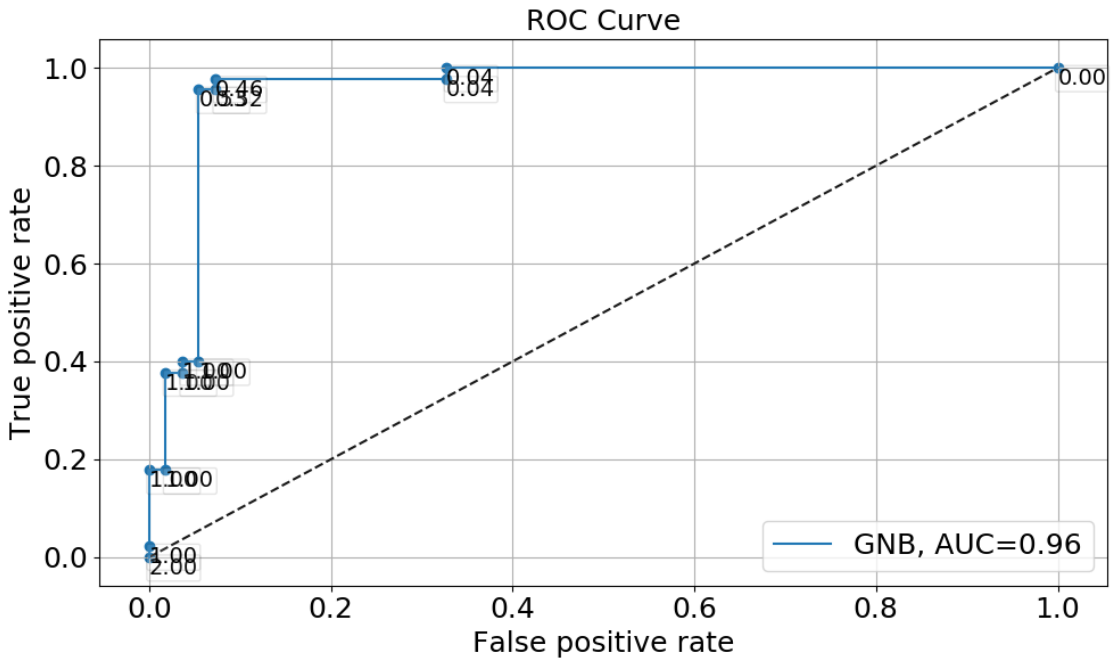
```
In [39]: print(ConfusionMatrix(y_test, y_pred_gnb))

Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100


In [40]: print(classification_report(y_test, y_pred_gnb, target_names=class_names))

              precision    recall  f1-score   support

           0       0.96      0.93      0.94        55
           1       0.91      0.96      0.93        45

   micro avg       0.94      0.94      0.94       100
   macro avg       0.94      0.94      0.94       100
weighted avg       0.94      0.94      0.94       100



In [41]: print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_gnb)))
         print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_gnb)))
         print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_gnb)))
         print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_gnb)))

Accuracy = 0.94
Kappa = 0.88
F1 Score = 0.93
Log Loss = 2.07
```

```
In [42]: from yellowbrick.classifier import DiscriminationThreshold

         visualizer = DiscriminationThreshold(gnb)
         visualizer.fit(X, y)  # Fit the training data to the visualizer
         visualizer.poof()     # Draw/show/poof the data
```

```
Out[42]: DiscriminationThreshold(argmax='fscore',
                     ax=<matplotlib.axes._subplots.AxesSubplot object at 0x00000246926BCBA8>,
                     cv=0.1, exclude=None, fbeta=1.0, model=None, n_trials=50,
                     quantiles=array([0.1, 0.5, 0.9]), random_state=None)
```

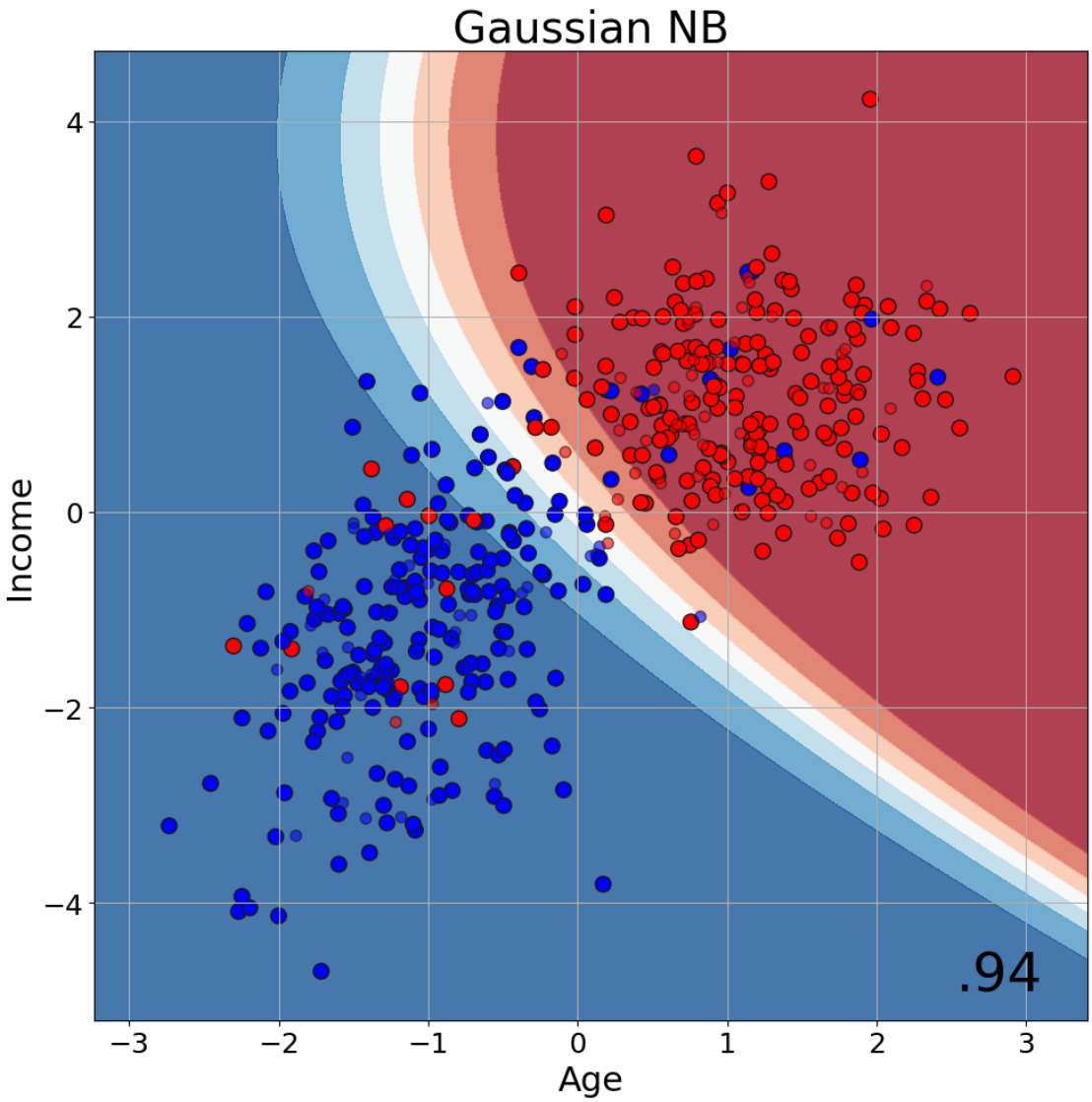



```
In [43]: plt.style.use('default');
         figure = plt.figure(figsize=(10, 6));
         ax = plt.subplot(1, 1, 1);
         plot_roc(gnb, X_test, y_test, "GNB", ax)
         plt.legend(loc='lower right', fontsize=18);
         plt.tight_layout();
         plt.savefig('out/default-gnb-roc.png');
```



7.3 Model Visualization

```
In [44]: figure = plt.figure(figsize=(10, 10));
         ax = plt.subplot(1, 1, 1);
         plot_boundaries(X_train, X_test, y_train, y_test, gnb, "Gaussian NB", ax, hide_ticks=
         ax.set_xlabel("Age", fontsize=22)
         ax.set_ylabel("Income", fontsize=22)
         plt.tight_layout();
         plt.savefig('out/default-gnb-boundaries.png');
```



8 KNN

```
In [45]: from sklearn.neighbors import KNeighborsClassifier

         knn_clf = KNeighborsClassifier(n_neighbors=3)
         knn_clf.fit(X_train, y_train)

         y_pred_knn = knn_clf.predict(X_test)
```

```
Out[45]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                    weights='uniform')
```

8.1 Model Parameters

```
In [46]: knn_clf.effective_metric_
         knn_clf.effective_metric_params_
```

```
Out[46]: 'euclidean'
```

```
Out[46]: {}
```

8.2 Model Performance

```
In [47]: print(ConfusionMatrix(y_test, y_pred_knn))
```

```
Predicted  False  True  __all__
Actual
False         47     8       55
True           2    43       45
__all__       49    51      100
```

```
In [48]: print(classification_report(y_test, y_pred_knn, target_names=class_names))
```

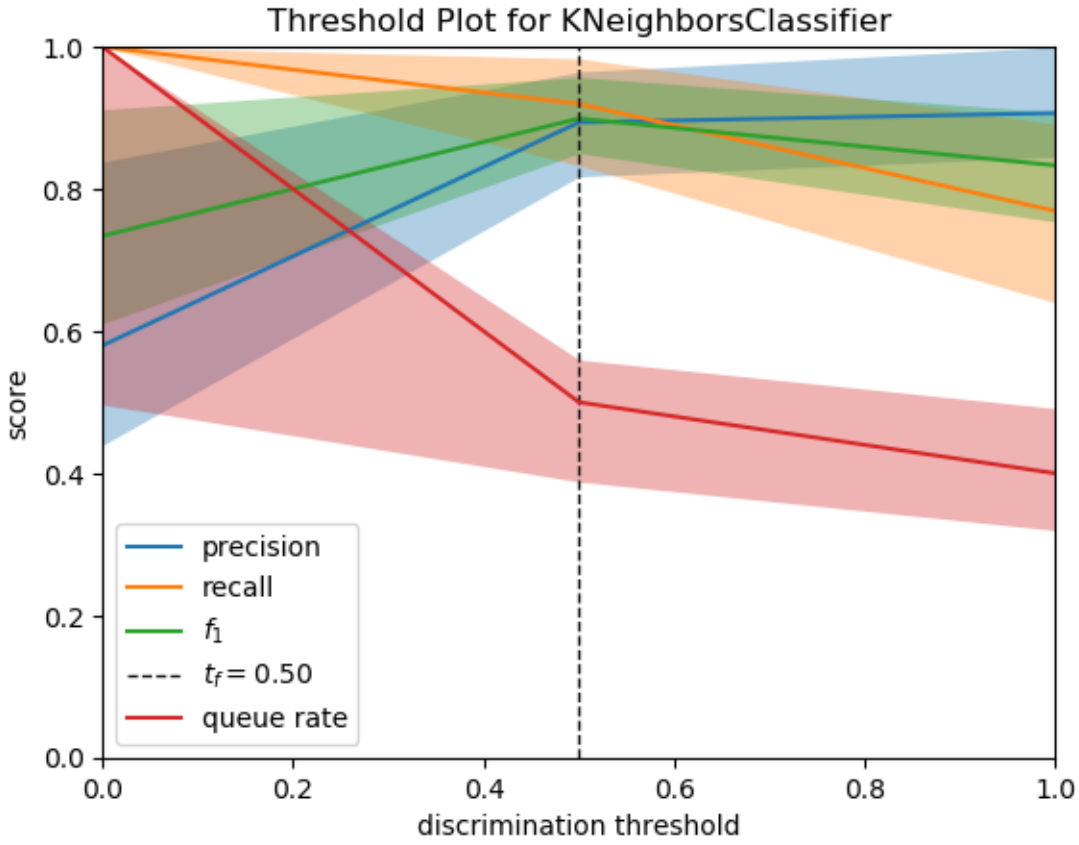
```
              precision    recall  f1-score   support

           0       0.96      0.85      0.90        55
           1       0.84      0.96      0.90        45

   micro avg       0.90      0.90      0.90       100
   macro avg       0.90      0.91      0.90       100
weighted avg       0.91      0.90      0.90       100
```

```
In [49]: print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_knn)))
         print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_knn)))
         print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_knn)))
         print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_knn)))
```

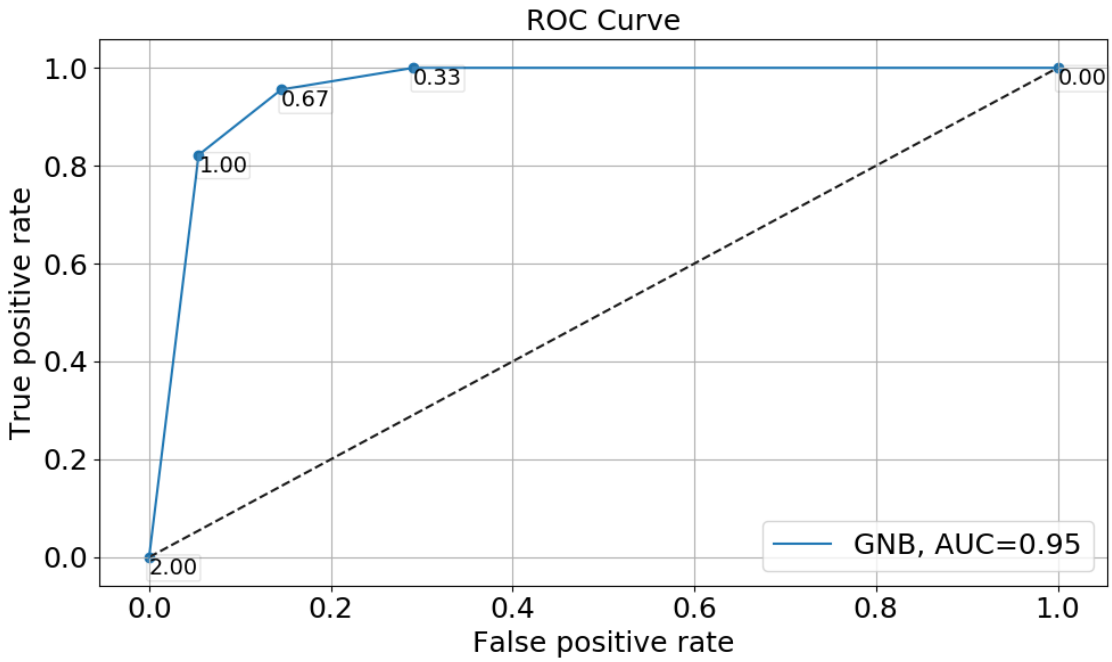
```
Accuracy = 0.90
Kappa = 0.80
F1 Score = 0.90
Log Loss = 3.45
```

```
In [50]: from yellowbrick.classifier import DiscriminationThreshold

         visualizer = DiscriminationThreshold(knn_clf)
         visualizer.fit(X, y)  # Fit the training data to the visualizer
         visualizer.poof()     # Draw/show/poof the data
```

Out[50]:
```
DiscriminationThreshold(argmax='fscore',
                        ax=<matplotlib.axes._subplots.AxesSubplot object at 0x00000246915CEB70>,
                        cv=0.1, exclude=None, fbeta=1.0, model=None, n_trials=50,
                        quantiles=array([0.1, 0.5, 0.9]), random_state=None)
```

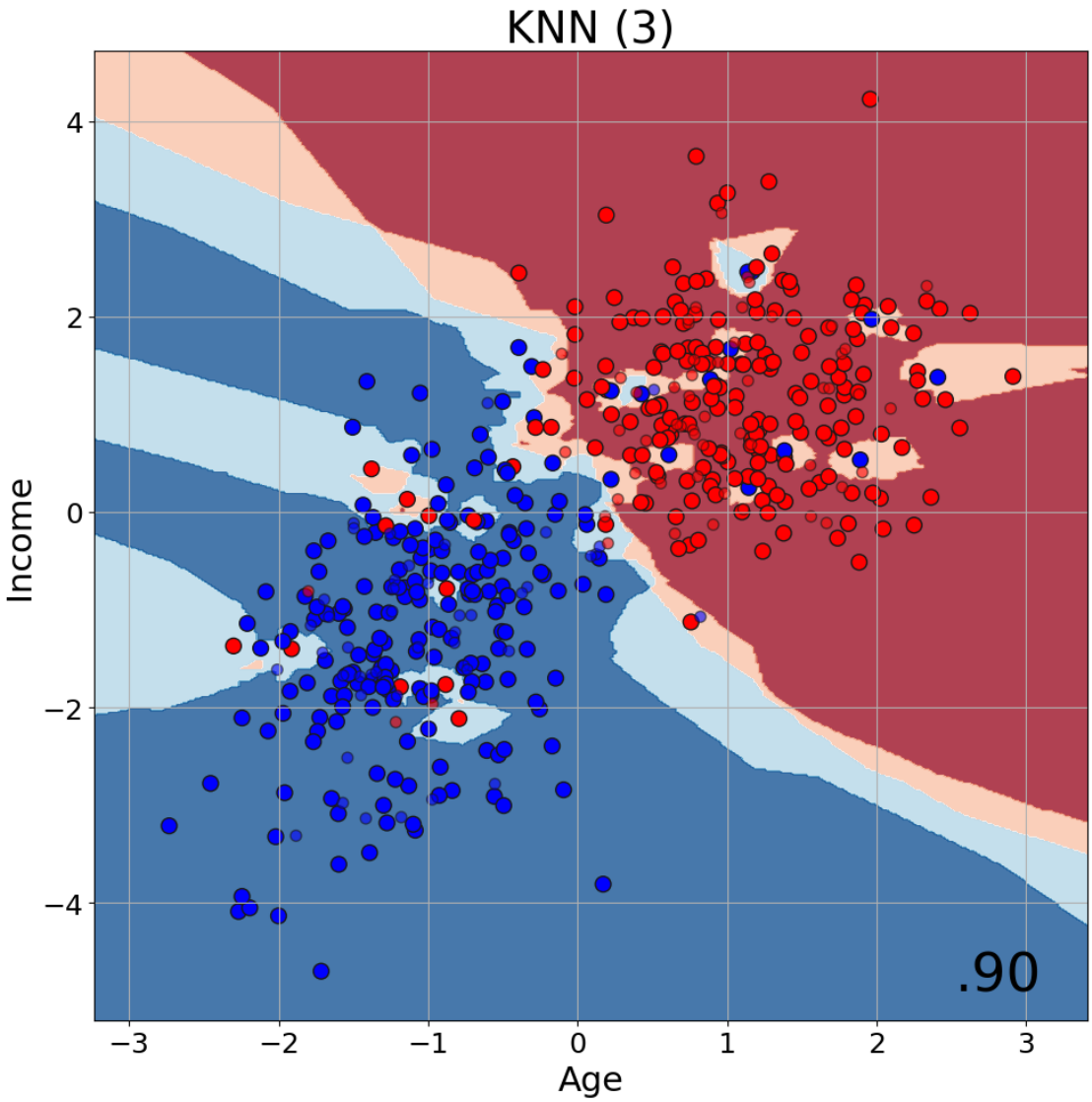


In [51]:
```
plt.style.use('default');
figure = plt.figure(figsize=(10, 6));
ax = plt.subplot(1, 1, 1);
plot_roc(knn_clf, X_test, y_test, "GNB", ax)
plt.legend(loc='lower right', fontsize=18);
plt.tight_layout();
plt.savefig('out/default-knn-roc.png');
```



8.3 Model Visualization

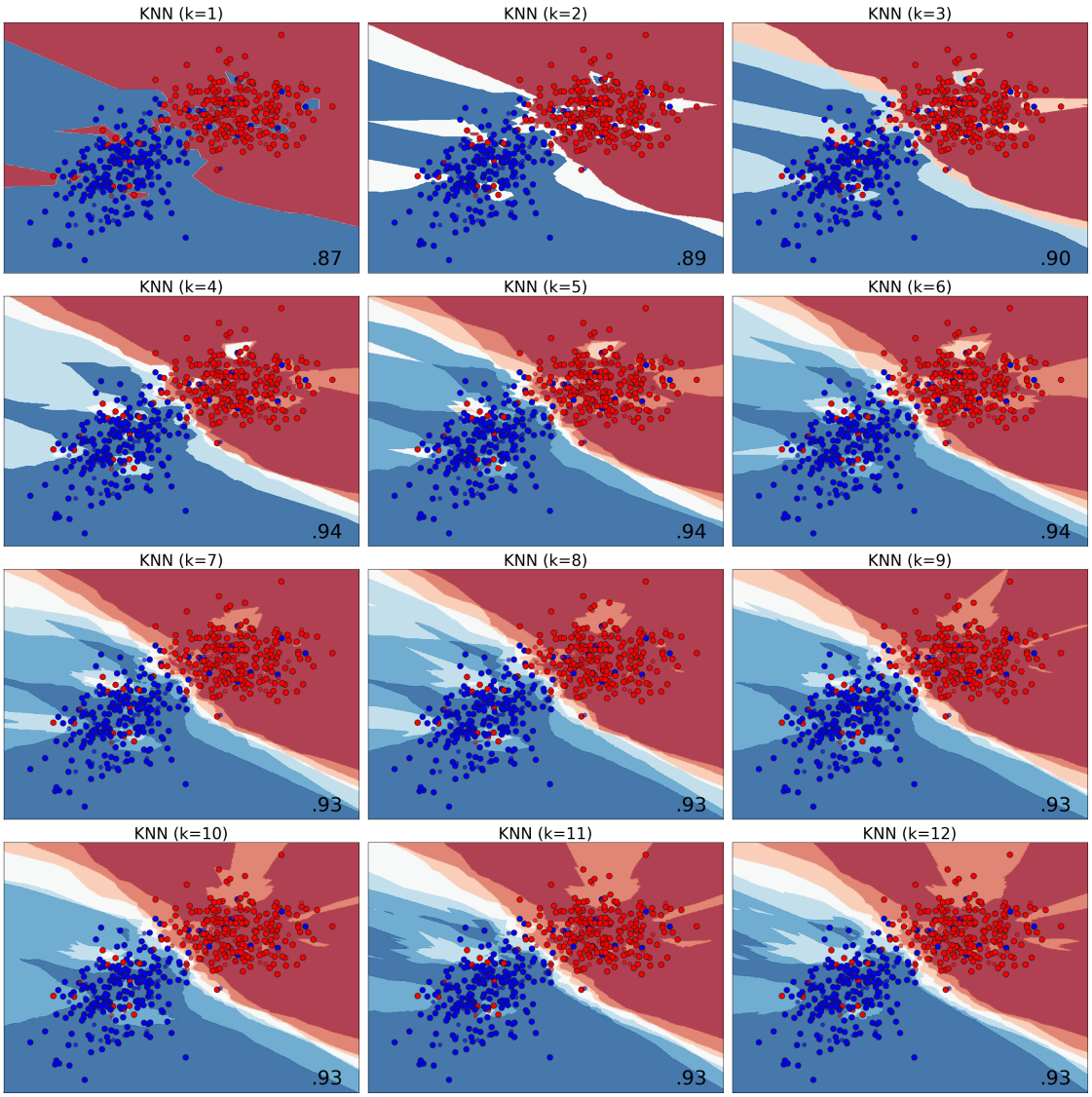
```
In [52]: figure = plt.figure(figsize=(10, 10));
         ax = plt.subplot(1, 1, 1);
         plot_boundaries(X_train, X_test, y_train, y_test, knn_clf, "KNN (3)", ax, hide_ticks=
         ax.set_xlabel("Age", fontsize=22)
         ax.set_ylabel("Income", fontsize=22)
         plt.tight_layout();
         plt.savefig('out/default-knn-boundaries.png');
```



8.4 Experimenting with Lots of Ks

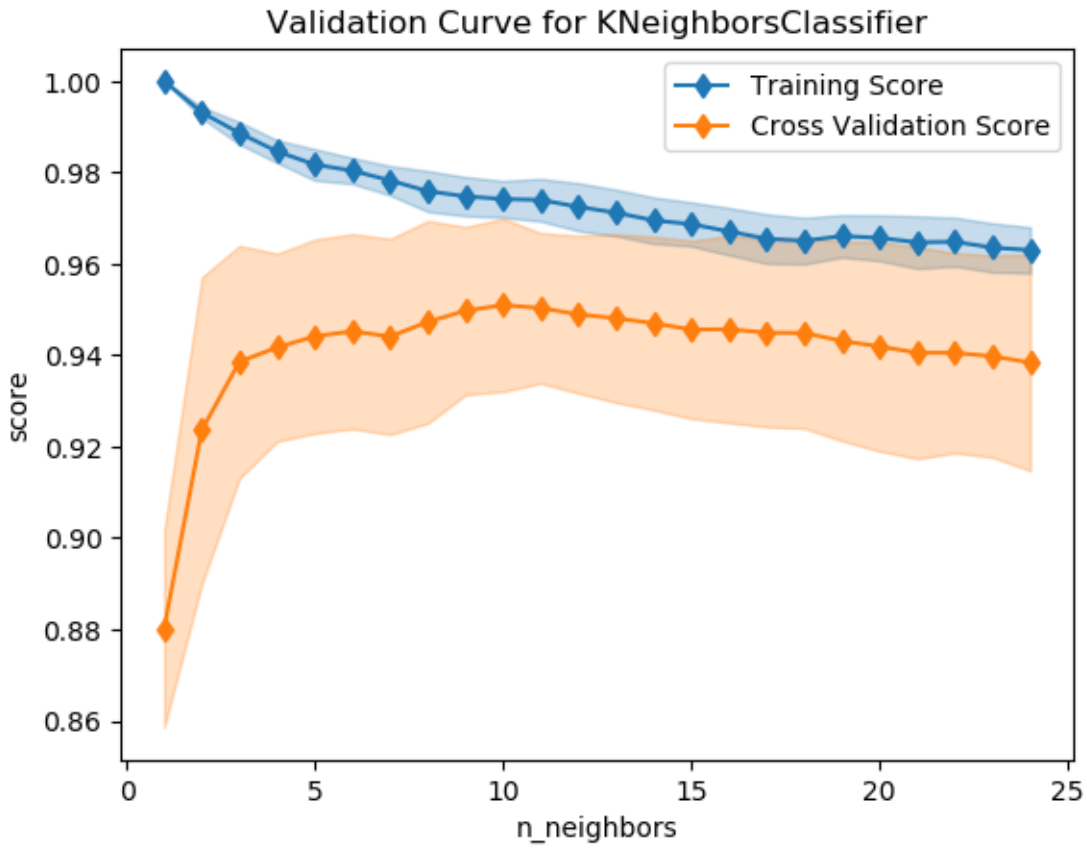
In [53]:
```
rng = np.random.RandomState(2)

figure = plt.figure(figsize=(27, 27));
i = 1

ks = np.arange(1, 13)

for k in ks:
    ax = plt.subplot(4, 3, i);
    clf_tmp = KNeighborsClassifier(n_neighbors=k)
    clf_tmp.fit(X_train, y_train);
```

```
        plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, "KNN (k={:d})".format(
        i += 1

    plt.tight_layout();
    plt.savefig('out/default-knn-all.png');
```



In [54]:
```
from yellowbrick.model_selection import ValidationCurve

viz = ValidationCurve(KNeighborsClassifier(), param_name="n_neighbors", param_range=np
viz.fit(X, y)
viz.poof(outpath='out/default-knn-validation.png')
viz.poof()
```

```
Out[54]: ValidationCurve(ax=<matplotlib.axes._subplots.AxesSubplot object at 0x000002469171343
             cv=5, groups=None, logx=False, model=None, n_jobs=1,
             param_name='n_neighbors',
             param_range=array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15
            18, 19, 20, 21, 22, 23, 24]),
             pre_dispatch='all', scoring='roc_auc')
```



9 SVM - Linear

```
In [55]: from sklearn.svm import SVC

         svm_clf = SVC(kernel="linear", C=0.025)
         svm_clf.fit(X_train, y_train)

         y_pred_svm = svm_clf.predict(X_test)
```

```
Out[55]: SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
           kernel='linear', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

9.1 Model Parameters

```
In [56]: svm_clf.n_support_

Out[56]: array([70, 69])

In [57]: svm_clf.support_vectors_

Out[57]: array([[-2.30188062e+00, -1.37169280e+00],
                [-1.18486685e+00, -1.79040549e+00],
                [ 6.59102878e-01, -4.92594995e-02],
                [ 1.88977492e-01,  1.49340633e+00],
                [ 6.32255404e-02,  1.15016778e+00],
                [ 1.23912826e+00, -4.01893060e-01],
                [ 7.49980152e-01, -3.38768128e-01],
                [ 1.23622216e+00,  1.18585351e-01],
                [ 7.19460491e-01,  9.02795935e-01],
                [ 1.95852933e-01,  1.24920912e+00],
                [ 6.24289241e-01,  7.92790019e-01],
                [ 6.76905028e-01, -3.75127816e-01],
                [ 9.03792039e-01,  3.30215452e-01],
                [ 1.27385324e+00, -1.12810817e-02],
                [-8.82249296e-01, -1.76640750e+00],
                [-2.19281611e-02,  1.37013900e+00],
                [ 6.18149343e-01,  8.72452885e-01],
                [ 7.33559474e-01,  3.18524739e-01],
                [ 5.64698788e-01,  8.86571007e-01],
                [-1.13932834e+00,  1.30141500e-01],
                [ 8.88865699e-01,  2.62250487e-01],
                [-1.37872953e+00,  4.40826598e-01],
                [ 8.41265426e-01,  4.54384311e-01],
                [ 1.14313600e+00,  3.62840642e-01],
                [ 9.24156804e-01,  1.75407970e-01],
                [ 7.58550586e-01, -1.12617319e+00],
                [ 5.37591993e-01,  3.67379991e-01],
                [ 5.10917745e-01,  1.47974842e+00],
                [-1.72530340e-02,  1.81742287e+00],
                [ 1.62179992e-01,  1.28041447e+00],
                [ 6.04206578e-01,  7.58038140e-01],
                [-1.28217990e+00, -1.40085298e-01],
                [ 3.54829980e-01,  5.80582909e-01],
                [ 7.65796356e-01,  1.13686808e-01],
                [ 4.77436289e-01,  1.06166759e+00],
                [ 7.44778555e-01,  9.00883919e-01],
                [ 4.54681092e-01,  8.98129139e-02],
                [-8.74340388e-01, -7.85845896e-01],
                [ 5.47430210e-01,  1.10098170e+00],
                [ 9.62642219e-01,  6.05067206e-01],
                [ 1.37891375e+00, -2.16938086e-01],
```

```
[-2.80853387e-01,  8.65743581e-01],
[ 1.18081729e-01,  6.57240174e-01],
[ 1.00579827e+00,  5.12674831e-01],
[ 1.05294772e+00,  3.40754580e-01],
[ 2.26064822e-01,  9.99447607e-01],
[ 5.28304707e-01,  4.04847493e-01],
[-1.73425327e-02,  2.10089864e+00],
[ 4.33733624e-01,  5.81012747e-01],
[ 9.68211922e-01,  5.73015154e-01],
[ 1.10018665e+00,  6.49157649e-04],
[ 5.52443998e-01,  7.34881333e-01],
[-3.92588186e-01,  2.44635394e+00],
[-4.32504847e-01,  4.63962927e-01],
[-1.91309520e+00, -1.40149499e+00],
[-1.75919852e-01,  8.67485179e-01],
[-9.94165067e-01, -3.83570608e-02],
[ 1.89138649e-01, -1.27882896e-01],
[ 8.07800717e-01, -2.89539100e-01],
[ 9.55808120e-01,  5.85645406e-01],
[-6.94505004e-01, -8.26621039e-02],
[-2.31687405e-01,  1.45868263e+00],
[ 5.50645183e-01,  1.09154900e+00],
[ 2.83255072e-01,  1.94337996e+00],
[ 6.21383822e-01,  9.61339062e-01],
[ 3.52555198e-01,  9.22969649e-01],
[-7.93173642e-01, -2.11572026e+00],
[ 4.24119906e-01,  9.69583682e-02],
[ 5.07250300e-01,  1.07466790e+00],
[ 8.78372008e-01,  6.48402453e-01],
[-3.57728806e-01, -9.72124403e-01],
[ 1.16945112e+00,  2.46206917e+00],
[ 1.02329442e+00,  1.66837945e+00],
[-8.54568112e-01, -1.08136079e-01],
[-6.10243208e-01, -9.25132872e-02],
[-6.51082450e-01,  7.93482409e-01],
[-2.90818242e-01,  9.66657192e-01],
[ 8.88877219e-01,  1.35711324e+00],
[ 1.14567439e+00,  2.49862993e-01],
[-6.72551783e-01, -1.02723989e-01],
[-1.65958857e-01,  5.01312397e-01],
[-1.11026103e+00,  5.80928590e-01],
[-1.46728331e-01, -1.70128704e+00],
[ 1.89174670e+00,  5.33504321e-01],
[ 2.23878764e-01,  3.33925832e-01],
[-3.29048799e-01, -4.21885955e-01],
[ 6.25696307e-02, -1.27624914e-01],
[-9.07768398e-01, -3.95867014e-01],
[ 1.45710938e-01, -4.72792669e-01],
```

```
[ 1.38446581e+00,  6.27550042e-01],
[ 3.65875063e-02, -7.40435348e-01],
[-5.96542747e-01,  5.60868593e-01],
[ 1.90456410e-01, -8.44946982e-01],
[-7.96003624e-01, -6.13619063e-01],
[ 1.13902431e+00,  2.45744539e+00],
[-3.93425323e-01,  1.68381711e+00],
[ 4.27929793e-01,  1.20813595e+00],
[-3.36699846e-01, -1.40488370e+00],
[-6.88371244e-01,  4.50933928e-01],
[-9.74212809e-01,  6.41743360e-01],
[-1.40930393e+00,  1.33588897e+00],
[-5.00902346e-01,  1.13145875e+00],
[-5.36700489e-01, -9.61351057e-01],
[ 5.14984518e-02, -2.62723318e-02],
[-4.55905745e-01, -2.09541216e-01],
[ 2.24847800e-01,  1.23928176e+00],
[-1.28788288e-01, -8.08351547e-01],
[-3.50507936e-01,  9.17139355e-02],
[-5.04218538e-01, -7.60450595e-01],
[-3.06737856e-01,  1.48961557e+00],
[-8.79079760e-01,  2.77804677e-01],
[-2.27924024e-01, -6.42300269e-01],
[-5.91781139e-01, -8.13169172e-01],
[-4.99326676e-01, -4.72936898e-01],
[-2.44865221e-01, -6.17301473e-01],
[-5.48039487e-01, -1.02064581e+00],
[-4.17773277e-01,  1.69670821e-01],
[-9.50700746e-01, -2.87491755e-01],
[-4.28889889e-01, -2.93609553e-01],
[-1.50383854e-01, -2.46748934e-02],
[-1.50502677e+00,  8.68032304e-01],
[-7.31758460e-01, -3.64788518e-02],
[-7.02561420e-01, -6.42527326e-01],
[-6.68197897e-01, -6.15539533e-01],
[-6.03591627e-01, -5.97638143e-01],
[-1.21659916e-01,  1.09059102e-01],
[-9.32723785e-01,  8.55382042e-02],
[ 1.96803236e+00,  1.97419278e+00],
[-3.41739912e-01, -1.70731937e-01],
[-5.83140024e-01, -4.94214194e-01],
[ 6.11267459e-01,  5.85651477e-01],
[-8.71380707e-01, -8.19669397e-02],
[-1.05262280e+00,  1.21705673e+00],
[ 2.41034306e+00,  1.38006683e+00],
[-4.57200965e-01, -2.31516041e-01],
[-4.84476078e-01,  4.31029381e-01],
[-6.62508962e-01, -4.09213248e-01],
```

```
        [-4.66745323e-01, -8.57262792e-01],
        [-4.68968326e-01,  3.99507234e-01]])
```

In [58]:
```
svm_clf.dual_coef_
```

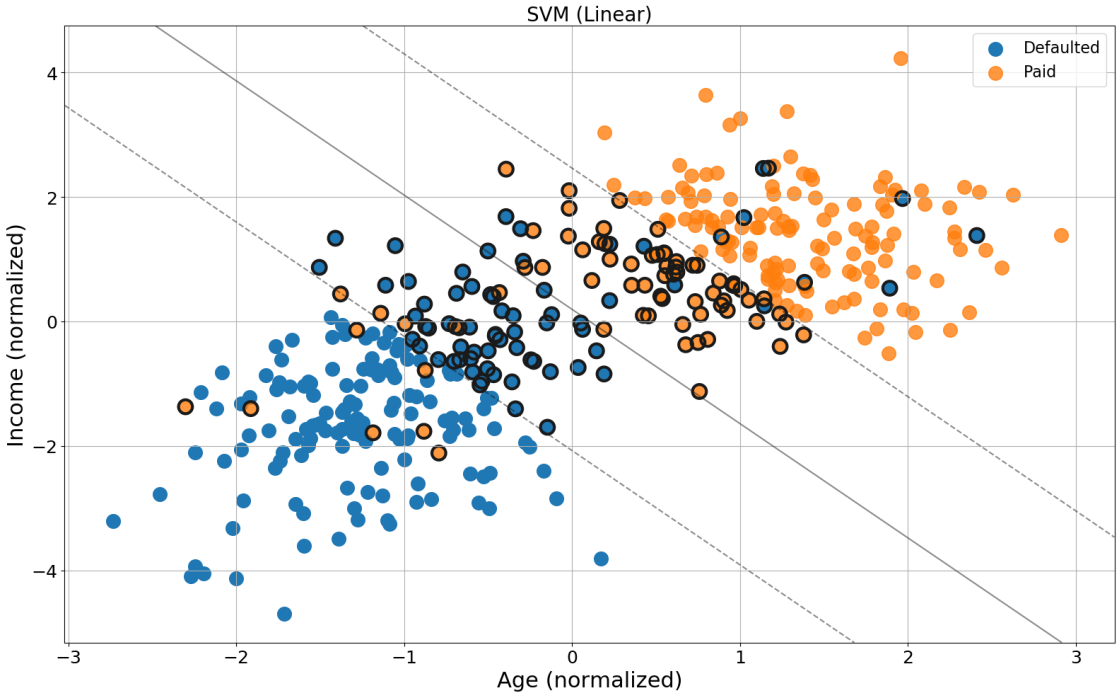
Out[58]:
```
array([[-0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.00055246, -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.01900603, -0.025     ,
        -0.025     , -0.025     , -0.025     , -0.025     , -0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.01955849,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ,  0.025     ,
         0.025     ,  0.025     ,  0.025     ,  0.025     ]])
```

In [59]:
```
svm_clf.intercept_
```

Out[59]:
```
array([0.0853027])
```

In [60]:
```
plt.figure(figsize=(16, 10));
plt.grid(True);

ind_d = y_train==1
ind_p = y_train==0
plt.scatter(X_train[ind_d,0], X_train[ind_d,1], marker='o', s=200, label='Defaulted')
plt.scatter(X_train[ind_p,0], X_train[ind_p,1], marker='o', s=200, label="Paid", alpha

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
```

```
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = svm_clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-'

# plot support vectors
plt.scatter(svm_clf.support_vectors_[:, 0], svm_clf.support_vectors_[:, 1], s=200, li

plt.legend(fontsize=16);
plt.title("SVM (Linear)", fontsize=20);
plt.xlabel('Age (normalized)', fontsize=22);
plt.ylabel('Income (normalized)', fontsize=22);
plt.xticks(fontsize=18);
plt.yticks(fontsize=18);
plt.tight_layout();
plt.savefig('out/default-svm-support-vectors.png');
```



9.2 Model Performance

```
In [61]: print(ConfusionMatrix(y_test, y_pred_svm))
```

```
Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100
```

```
In [62]: print(classification_report(y_test, y_pred_svm, target_names=class_names))
```

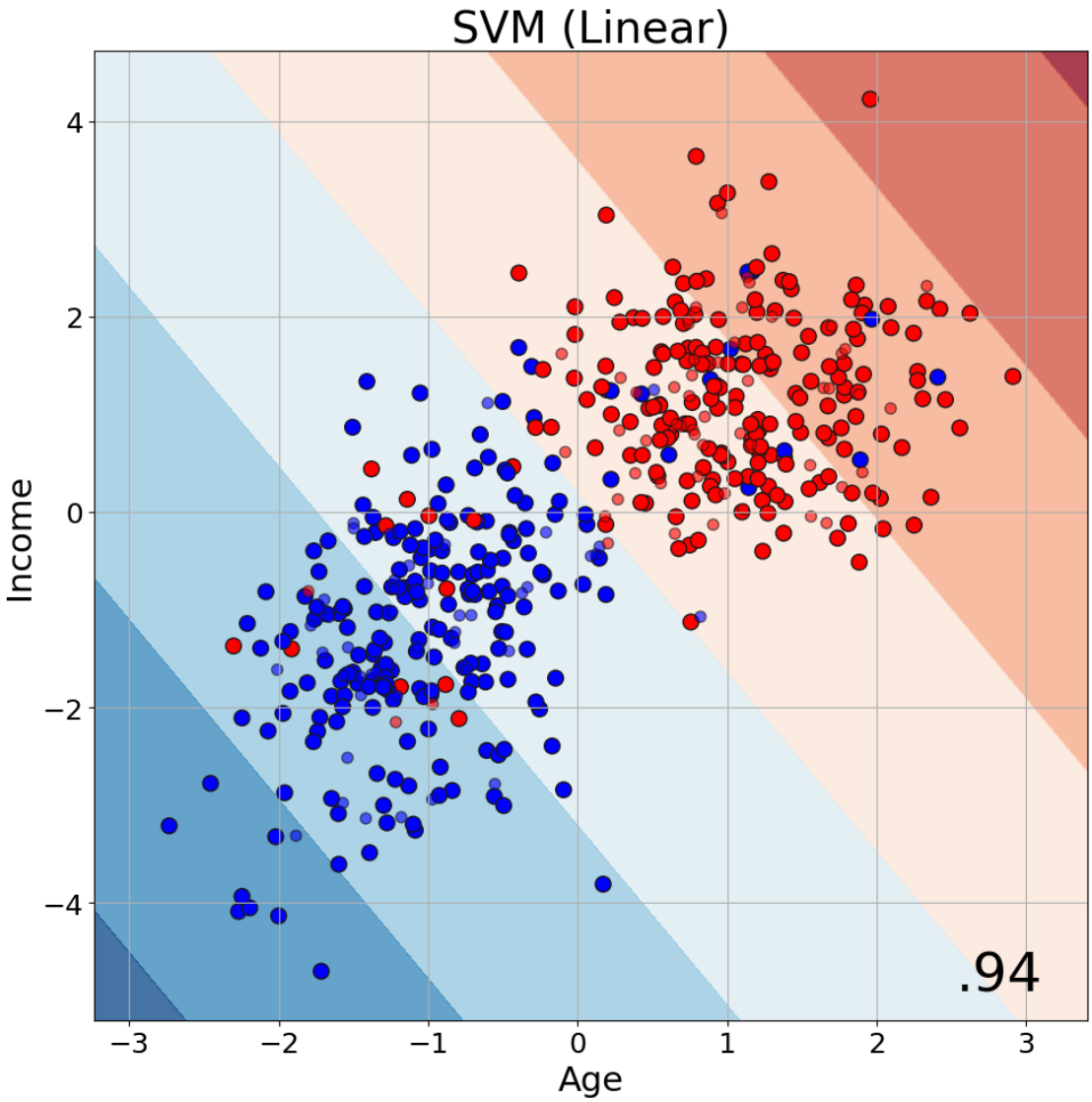
```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94        55
           1       0.91      0.96      0.93        45

   micro avg       0.94      0.94      0.94       100
   macro avg       0.94      0.94      0.94       100
weighted avg       0.94      0.94      0.94       100
```

```
In [63]: print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_svm)))
         print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_svm)))
         print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_svm)))
         print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_svm)))
```

```
Accuracy = 0.94
Kappa = 0.88
F1 Score = 0.93
Log Loss = 2.07
```

9.3 Model Visualization

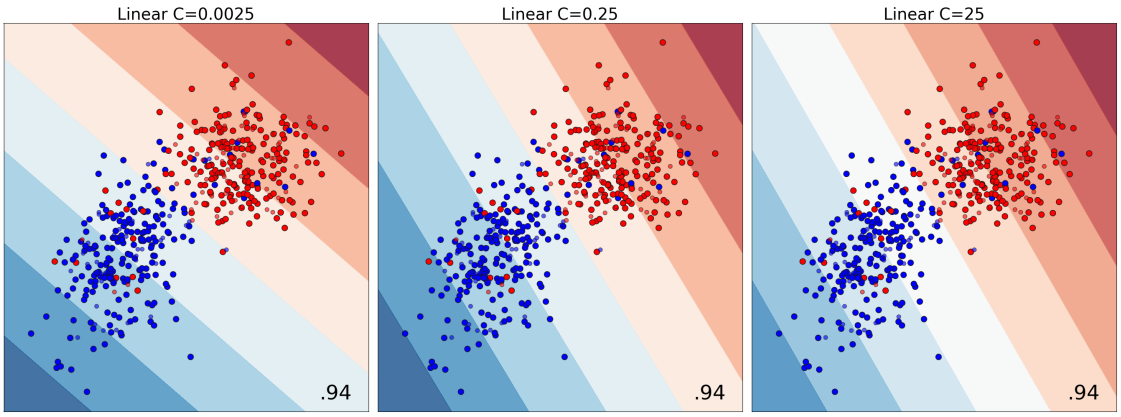
```
In [64]: figure = plt.figure(figsize=(10, 10));
         ax = plt.subplot(1, 1, 1);
         plot_boundaries(X_train, X_test, y_train, y_test, svm_clf, "SVM (Linear)", ax, hide_t
         ax.set_xlabel("Age", fontsize=22)
         ax.set_ylabel("Income", fontsize=22)
         plt.tight_layout();
         plt.savefig('out/default-svm-linear-boundaries.png');
```



9.4 Experimenting with Different Kernels

```
In [65]: names = ["Linear C=0.0025", "Linear C=0.25", "Linear C=25"]

         classifiers = [
             SVC(kernel="linear", C=0.0025),
             SVC(kernel="linear", C=0.25),
             SVC(kernel="linear", C=25),
         ]

         rng = np.random.RandomState(2)

         figure = plt.figure(figsize=(27, 10));
```

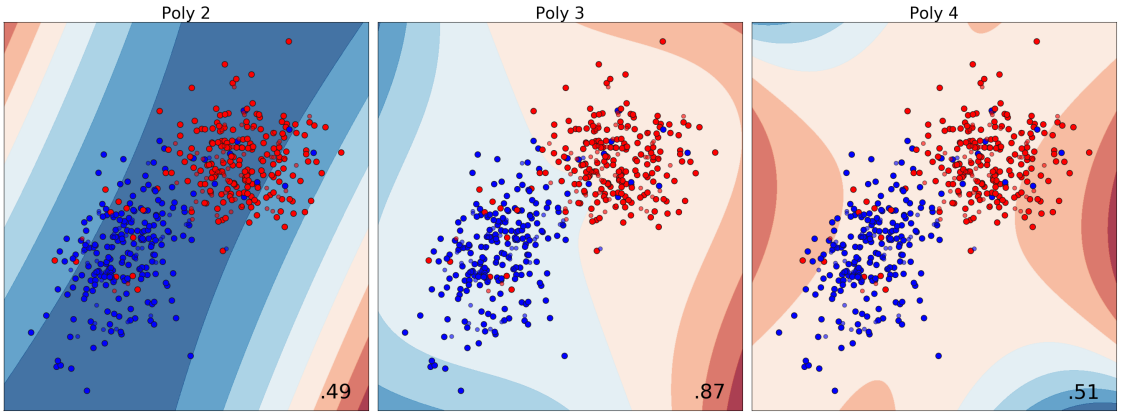
```
i = 1

# iterate over classifiers
for name, clf_tmp in zip(names, classifiers):
    ax = plt.subplot(1, 3, i);
    clf_tmp.fit(X_train, y_train);
    plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, name, ax, hide_ticks=Tr
    i += 1

plt.tight_layout();
plt.savefig('out/default-svm-linear-all.png');
```



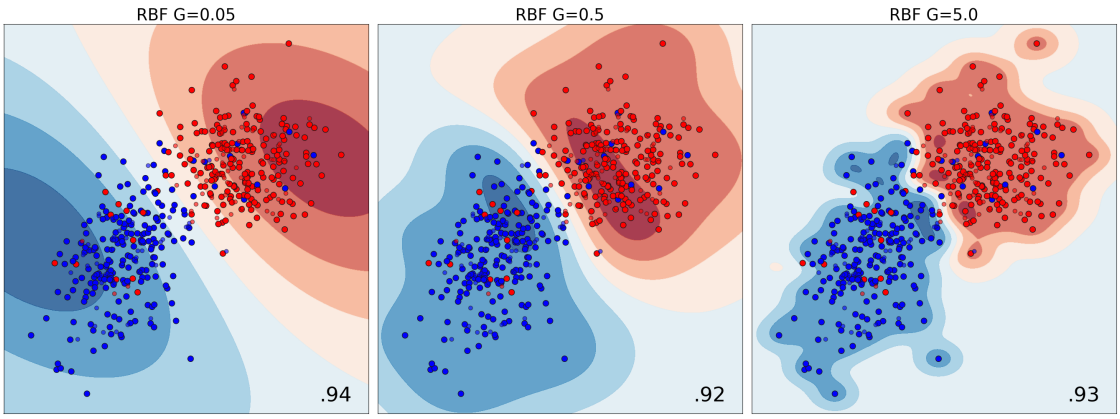

```
In [66]: names = ["Poly 2", "Poly 3", "Poly 4"]

classifiers = [
    SVC(kernel="poly", degree=2, C=0.25),
    SVC(kernel="poly", degree=3, C=1),
    SVC(kernel="poly", degree=4, C=1),
]

rng = np.random.RandomState(2)

figure = plt.figure(figsize=(27, 10));
i = 1

# iterate over classifiers
for name, clf_tmp in zip(names, classifiers):
    ax = plt.subplot(1, 3, i);
    clf_tmp.fit(X_train, y_train);
    plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, name, ax, hide_ticks=Tr
    i += 1
```

```
        plt.tight_layout();
        plt.savefig('out/default-svm-poly-all.png');
```

```
C:\Users\st50\AppData\Local\Continuum\anaconda3\envs\small_sklearn\lib\site-packages\sklearn\sv
  "avoid this warning.", FutureWarning)
C:\Users\st50\AppData\Local\Continuum\anaconda3\envs\small_sklearn\lib\site-packages\sklearn\sv
  "avoid this warning.", FutureWarning)
C:\Users\st50\AppData\Local\Continuum\anaconda3\envs\small_sklearn\lib\site-packages\sklearn\sv
  "avoid this warning.", FutureWarning)
```



In [67]:
```
names = ["RBF G=0.05", "RBF G=0.5", "RBF G=5.0"]

        classifiers = [
            SVC(kernel="rbf", gamma=0.05, C=1),
            SVC(kernel="rbf", gamma=0.5, C=1),
            SVC(kernel="rbf", gamma=5.0, C=1),
        ]

        rng = np.random.RandomState(2)

        figure = plt.figure(figsize=(27, 10));
        i = 1

        # iterate over classifiers
        for name, clf_tmp in zip(names, classifiers):
            ax = plt.subplot(1, 3, i);
            clf_tmp.fit(X_train, y_train);
            plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, name, ax, hide_ticks=Tr
            i += 1

        plt.tight_layout();
        plt.savefig('out/default-svm-rbf-all.png');
```



10 NN

```
In [68]: from sklearn.neural_network import MLPClassifier

         nn_clf = MLPClassifier(solver='lbfgs', activation='relu', alpha=1e-3,
                                hidden_layer_sizes=(3), random_state=1, verbose=True)
         nn_clf.fit(X_train, y_train)

         y_pred_nn = nn_clf.predict(X_test)
```

```
Out[68]: MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=3, learning_rate='constant',
                learning_rate_init=0.001, max_iter=200, momentum=0.9,
                n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                random_state=1, shuffle=True, solver='lbfgs', tol=0.0001,
                validation_fraction=0.1, verbose=True, warm_start=False)
```

10.1 Model Parameters

```
In [69]: nn_clf.loss_
```

```
Out[69]: 0.27106495217711907
```

```
In [70]: nn_clf.n_layers_
```

```
Out[70]: 3
```

```
In [71]: w = nn_clf.coefs_ # The ith element in the list represents the weight matrix correspo
         w
```

```
Out[71]: [array([[ 0.9756526 ,  1.08832583, -1.04406553],
                 [-1.66548065, -1.62569205, -0.61746919]]), array([[ 3.91733101],
                 [-2.57062869],
                 [ 1.76672827]])]
```

```
In [72]: b = nn_clf.intercepts_ # The ith element in the list represents the bias vector corre
         b

Out[72]: [array([-2.08114865, -1.19151499,  1.62527184]), array([-2.59817514])]

In [73]: nn_clf.out_activation_

Out[73]: 'logistic'

In [74]: nn_clf.predict_proba([[0.5, 1]])

Out[74]: array([[0.85068094, 0.14931906]])

In [75]: def relu_af(x):
             return max(0, x)

         x = [0.5, 1]

         layer_id=0
         h0 = (x[0] * w[layer_id][0][0]) + ((x[1] * w[layer_id][1][0])) + (1 * b[layer_id][0])
         print(h0)
         print(relu_af(h0))

         h1 = (x[0] * w[layer_id][0][1]) + ((x[1] * w[layer_id][1][1])) + (1 * b[layer_id][1])
         print(h1)
         print(relu_af(h1))

         h2 = (x[0] * w[layer_id][0][2]) + ((x[1] * w[layer_id][1][2])) + (1 * b[layer_id][2])
         print(h2)
         print(relu_af(h2))

         layer_id=1
         o = (h0 * w[layer_id][0][0]) + ((h1 * w[layer_id][1][0])) + ((h2 * w[layer_id][2][0]))
         print(o)
         print(relu_af(o))

-3.2588029971077006
0
-2.2730441233420393
0
0.48576987738849486
0.48576987738849486
-8.662609371944365
0
```

10.2 Model Performance

```
In [76]: print(ConfusionMatrix(y_test, y_pred_nn))
```

```
Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100
```

```
In [77]: print(classification_report(y_test, y_pred_nn, target_names=class_names))
```

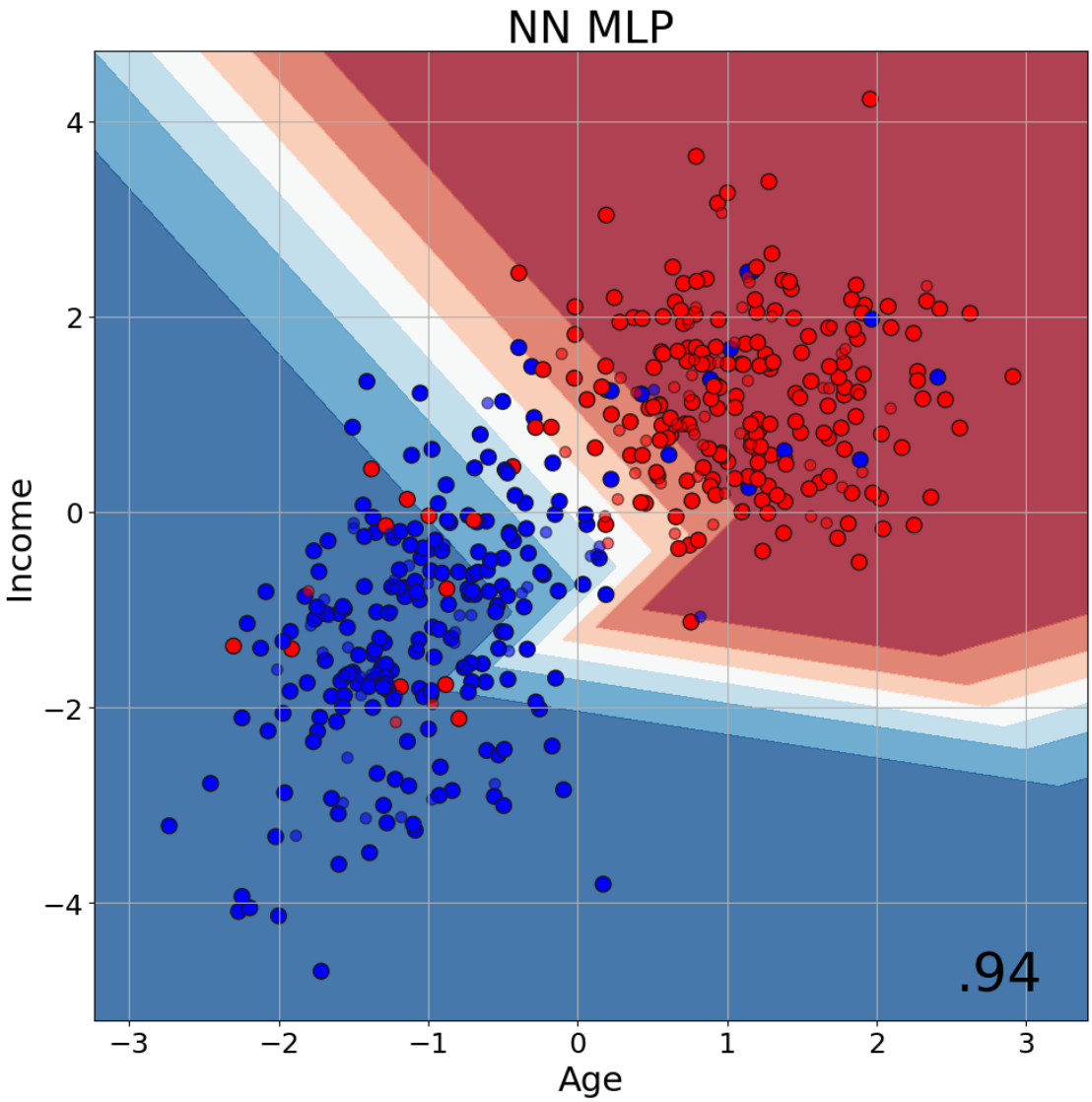
```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94        55
           1       0.91      0.96      0.93        45

   micro avg       0.94      0.94      0.94       100
   macro avg       0.94      0.94      0.94       100
weighted avg       0.94      0.94      0.94       100
```

```
In [78]: print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_nn)))
         print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_nn)))
         print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_nn)))
         print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_nn)))
```

```
Accuracy = 0.94
Kappa = 0.88
F1 Score = 0.93
Log Loss = 2.07
```

10.3 Model Visualization

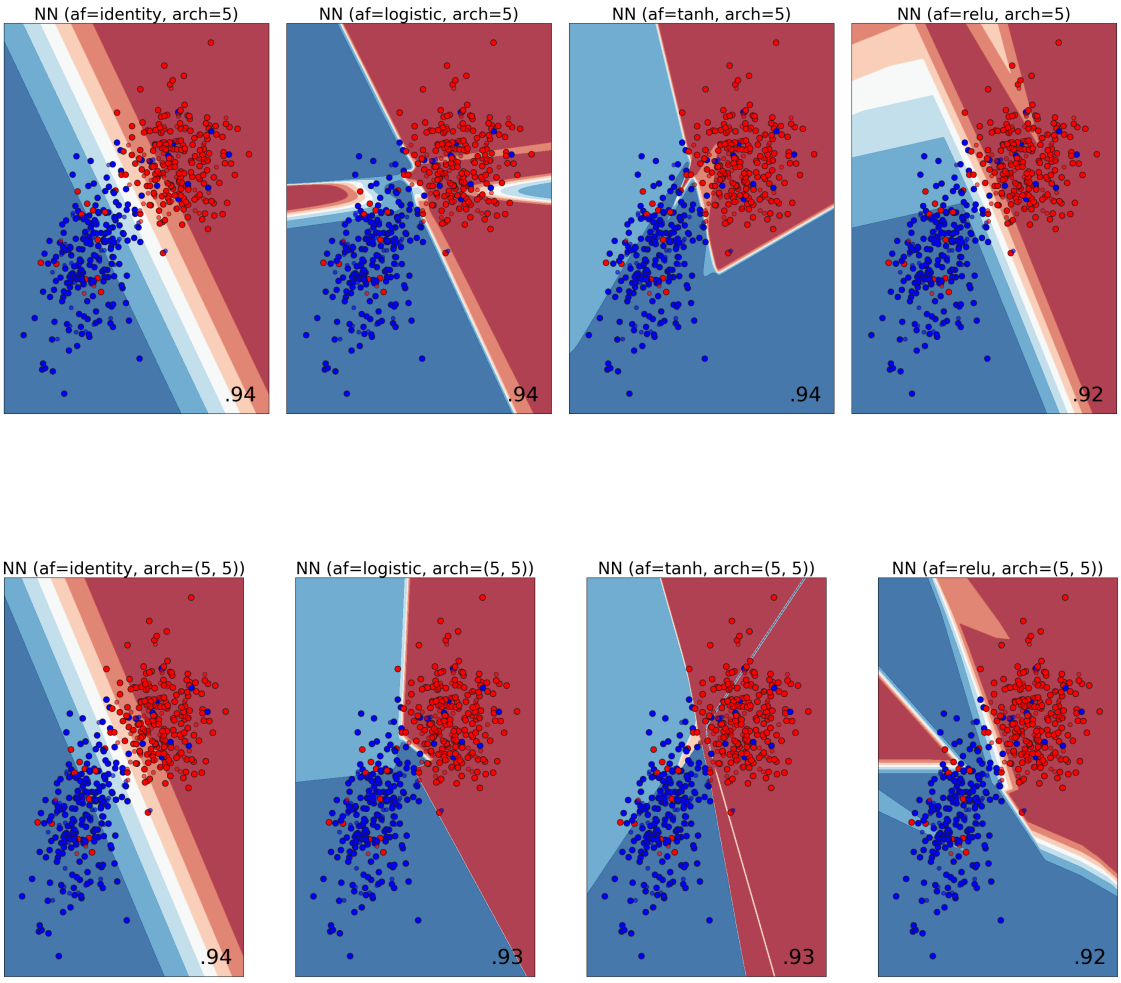
```
In [79]: figure = plt.figure(figsize=(10, 10));
         ax = plt.subplot(1, 1, 1);
         plot_boundaries(X_train, X_test, y_train, y_test, nn_clf, "NN MLP", ax, hide_ticks=Fa
         ax.set_xlabel("Age", fontsize=22)
         ax.set_ylabel("Income", fontsize=22)
         plt.tight_layout();
         plt.savefig('out/default-nn-mlp-boundaries.png');
```

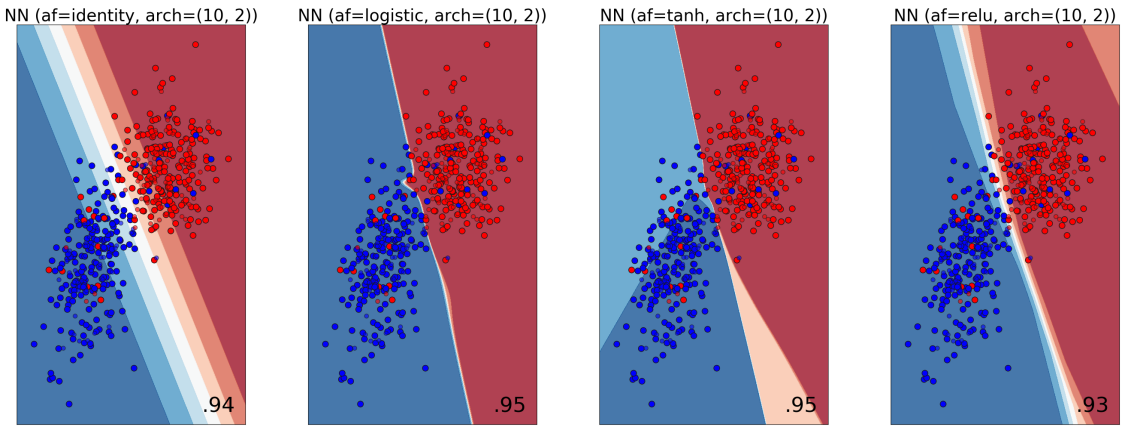


10.4 Experimenting with Mulitple Activation Functions

```
In [80]: rng = np.random.RandomState(2)

         alpha=0.001
         archs = [(5), (5, 5), (10, 2)]
         afs = ['identity', 'logistic', 'tanh', 'relu']

         j = 1

         for arch in archs:
             figure = plt.figure(figsize=(27, 10));
             i = 1
```

```
for af in afs:
    ax = plt.subplot(1, 4, i);
    clf_tmp = MLPClassifier(solver='lbfgs', activation=af, alpha=alpha, hidden_lay
    clf_tmp = clf_tmp.fit(X_train, y_train);
    plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, "NN (af={}, arch={
    i += 1

plt.tight_layout();
plt.savefig('out/default-nn-arch-{:d}-af.png'.format(j));
j += 1
```

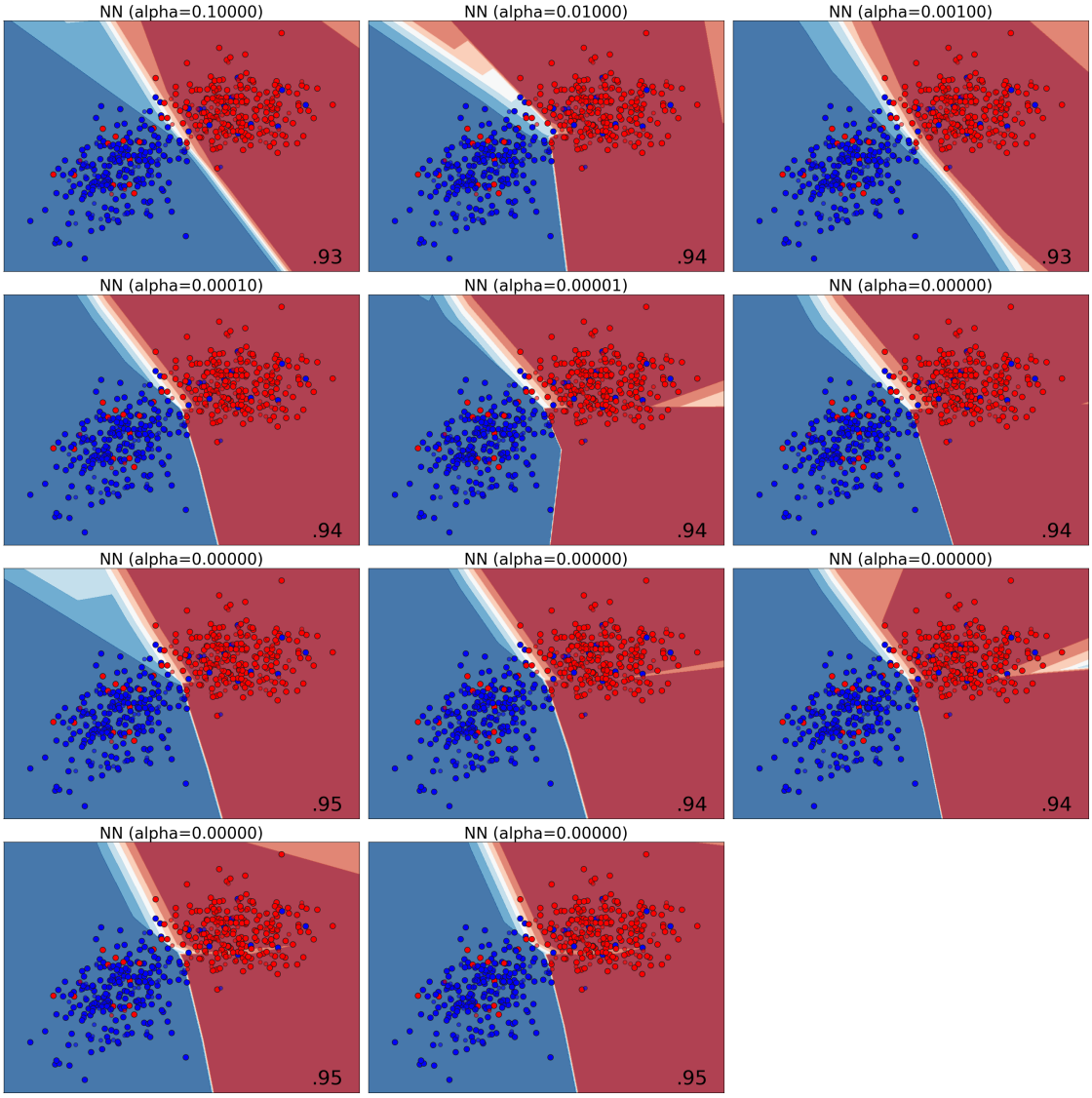




10.5 Experimenting with Mulitple Alphas

In [81]:
```
rng = np.random.RandomState(2)

figure = plt.figure(figsize=(27, 27));
i = 1

alphas = 10.0 ** -np.arange(1, 12)
print(alphas)

for alpha in alphas:
    ax = plt.subplot(4, 3, i);
    clf_tmp = MLPClassifier(solver='lbfgs', activation='relu', alpha=alpha, hidden_lay
    clf_tmp.fit(X_train, y_train);
    plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, "NN (alpha={:.5f})".for
    i += 1

plt.tight_layout();
plt.savefig('out/default-nn-all.png');
```

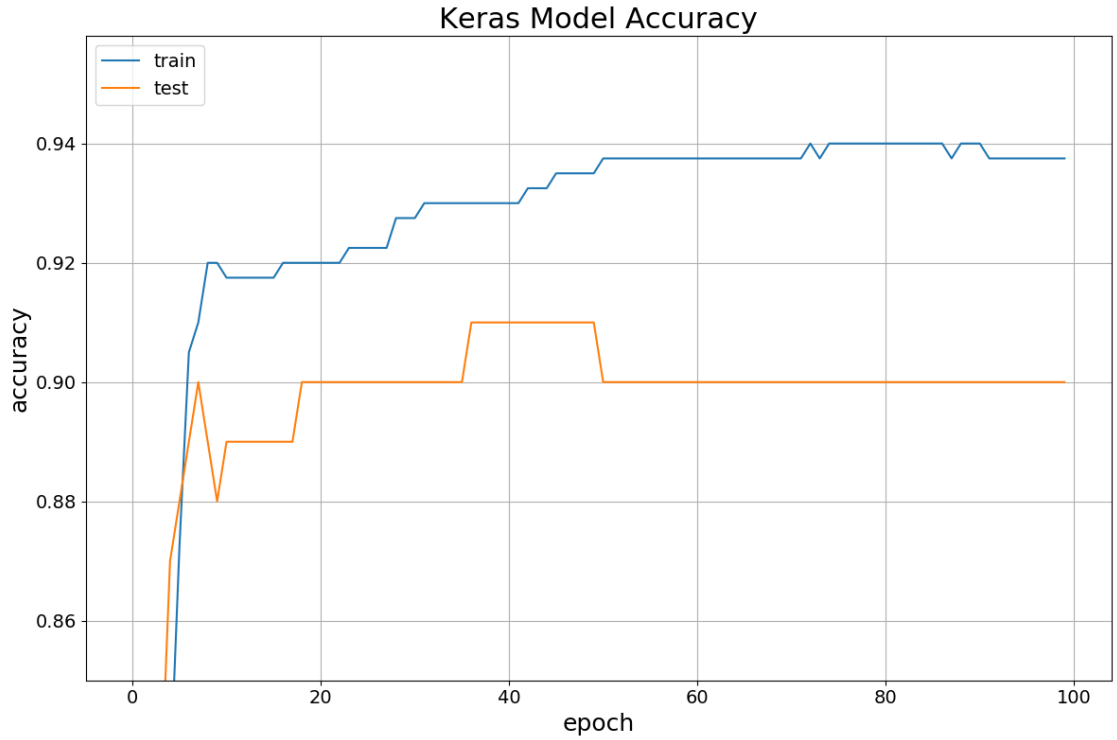
```
[1.e-01 1.e-02 1.e-03 1.e-04 1.e-05 1.e-06 1.e-07 1.e-08 1.e-09 1.e-10
 1.e-11]
```



11 NN - Keras

```
In [82]: from keras.models import Sequential
         from keras.layers import Dense
         from keras.utils import np_utils

         model = Sequential()
         model.add(Dense(3, input_dim=2, activation="relu"))
         model.add(Dense(1, activation='sigmoid'))
         model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

         %time history = model.fit(X, y, validation_split=0.20, epochs=100, batch_size=10, verb
```

```
Using TensorFlow backend.


Wall time: 3.78 s


In [83]: # list all data in history
         print(history.history.keys())

         figure = plt.figure(figsize=(12, 8));

         plt.plot(history.history['acc']);
         plt.plot(history.history['val_acc']);
         plt.title('Keras Model Accuracy', fontsize=22);
         plt.ylabel('accuracy', fontsize=18);
         plt.xlabel('epoch', fontsize=18);
         plt.xticks(fontsize=14);
         plt.yticks(fontsize=14);
         plt.ylim(bottom=0.85);
         plt.legend(['train', 'test'], loc='upper left', fontsize=14);

         plt.grid();
         plt.tight_layout();
         plt.savefig('out/default-keras-accuracy.png');

dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



In [84]:
```
# calculate predictions
predictions = model.predict(X_test)
# round predictions
y_pred_keras = [round(x[0]) for x in predictions]
print(ConfusionMatrix(y_test, y_pred_keras))
```

```
Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100
```

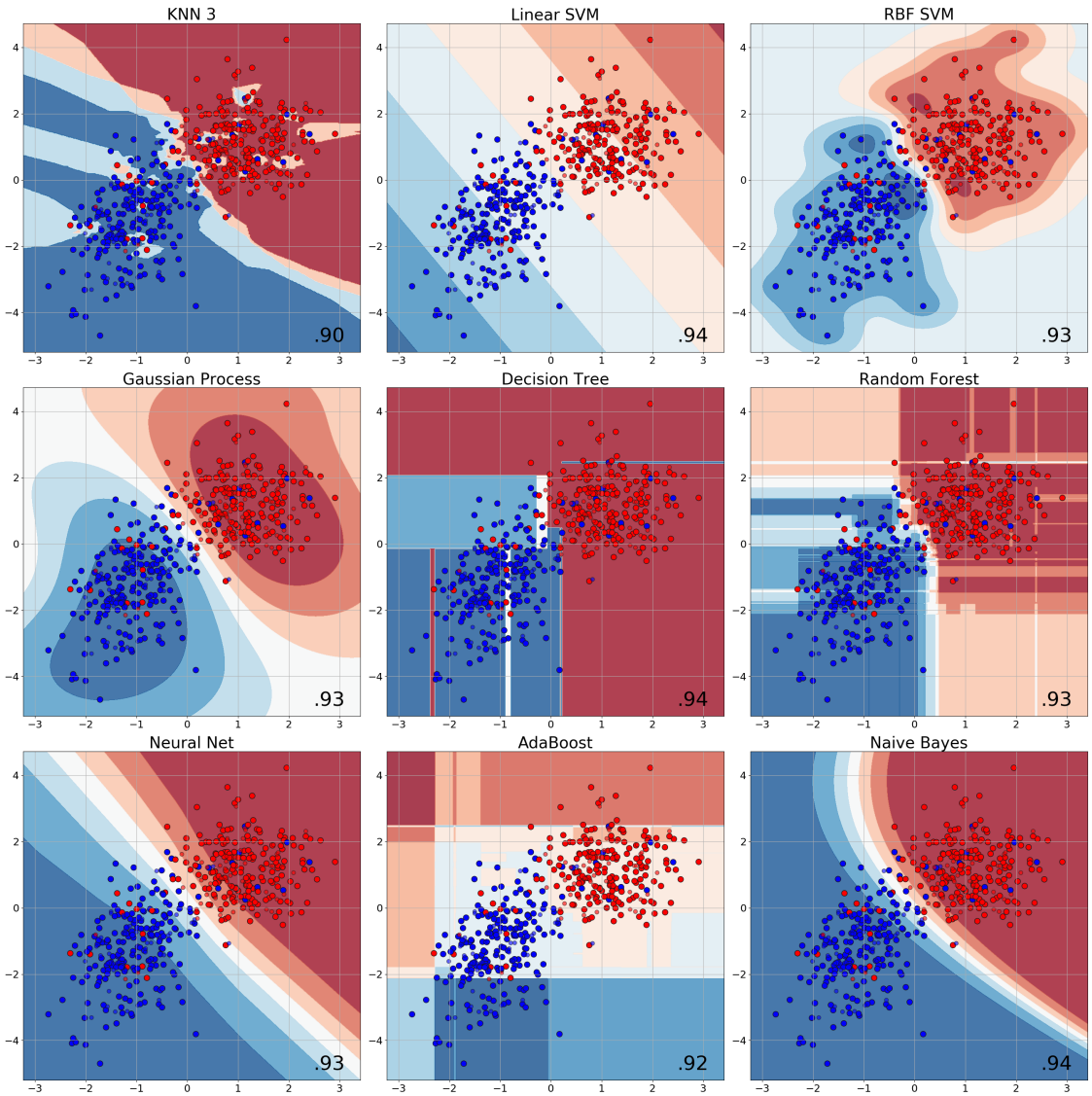
In [86]:
```
#from keras.utils import plot_model

#plot_model(model, to_file='out/default-keras-model.png', show_shapes=True, show_laye
```

In [87]:
```
#from ann_visualizer.visualize import ann_viz

#ann_viz(model, view=True, filename='out/default-keras-ann.gv', title="Default Datase
```

12 Comparing Multiple Algorithms

In [88]:
```
# Inspired by: https://scikit-learn.org/stable/auto_examples/classification/plot_clas

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis


names = ["KNN 3", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes"]
```

```
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1),
    AdaBoostClassifier(),
    GaussianNB()]


rng = np.random.RandomState(2)

figure = plt.figure(figsize=(27, 27));
i = 1

# iterate over classifiers
for name, clf_tmp in zip(names, classifiers):
    ax = plt.subplot(3, 3, i);
    clf_tmp.fit(X_train, y_train);
    plot_boundaries(X_train, X_test, y_train, y_test, clf_tmp, name, ax, hide_ticks=Fa
    i += 1

plt.tight_layout();
plt.savefig('out/default-all.png');
```


KNN 3
.90
Linear SVM
.94
RBF SVM
.93
Gaussian Process
.93
Decision Tree
.94
Random Forest
.93
Neural Net
.93
AdaBoost
.92
Naive Bayes
.94