---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-machine-learning/resources/bANLa) course resource.*

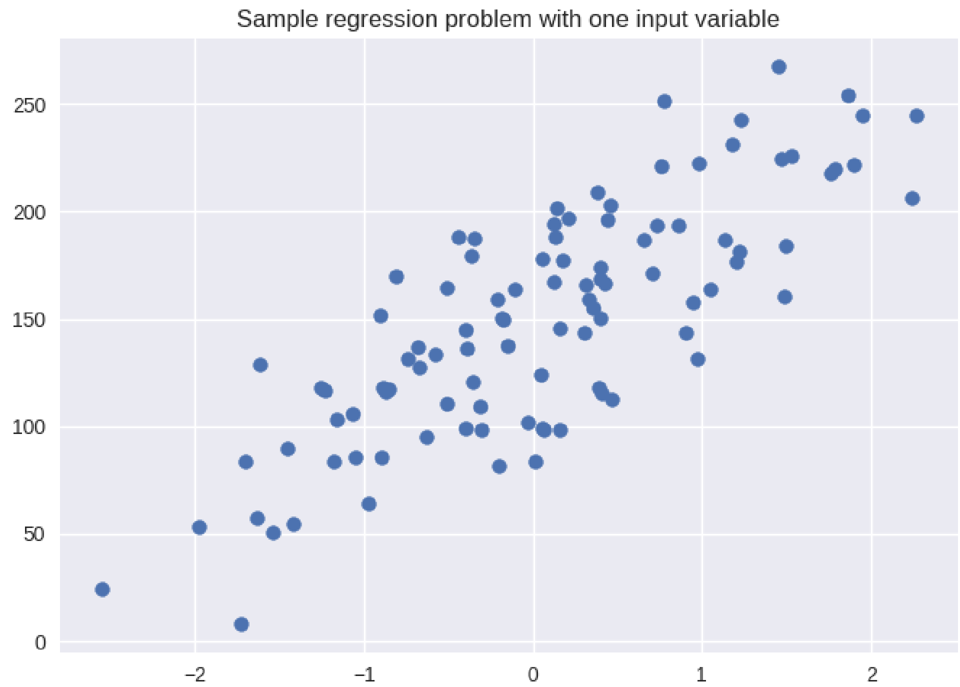
Applied Machine Learning: Module 4 (Supervised Learning, Part II) ¶

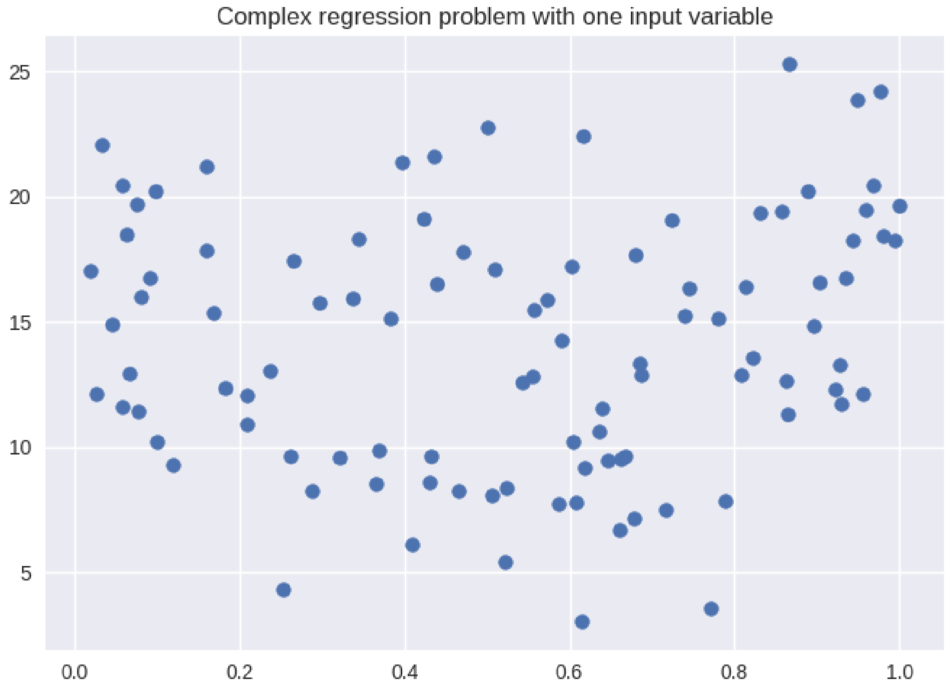
Preamble and Datasets

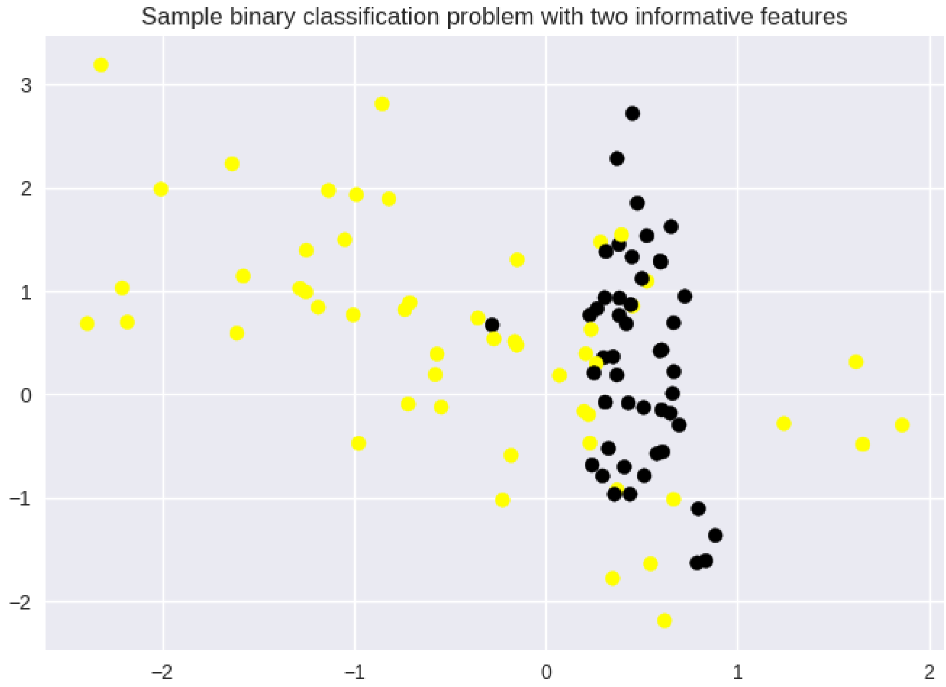
In [2]:

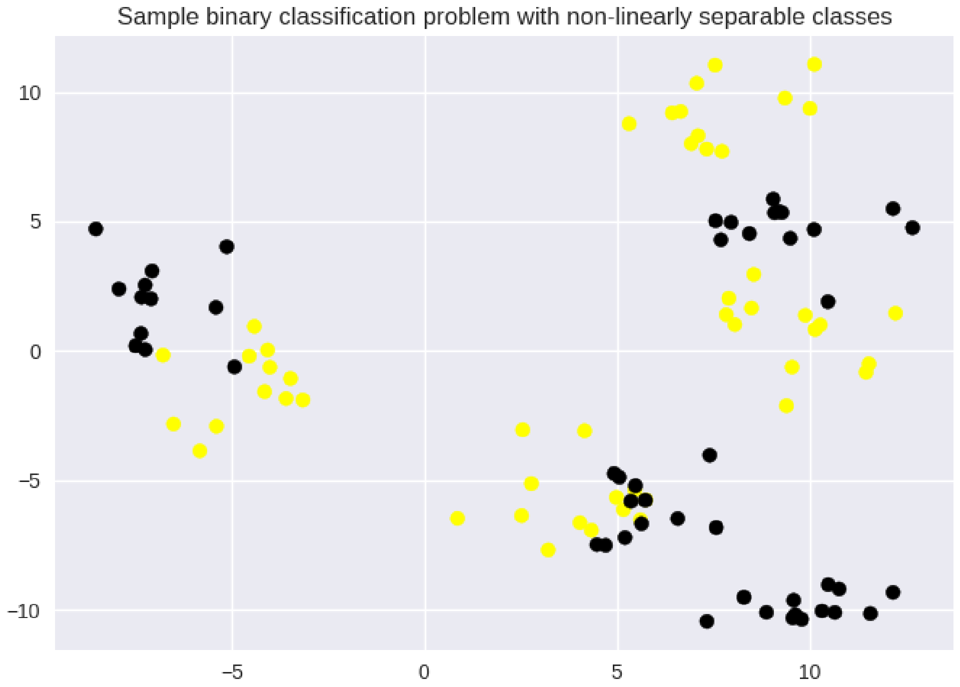
```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification, make_blobs
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_breast_cancer
from adspy_shared_utilities import load_crime_dataset


cmap_bold = ListedColormap(['#FFFF00', '#00FF00', '#0000FF','#000000'])

# fruits dataset
fruits = pd.read_table('readonly/fruit_data_with_colors.txt')

feature_names_fruits = ['height', 'width', 'mass', 'color_score']
X_fruits = fruits[feature_names_fruits]
y_fruits = fruits['fruit_label']
target_names_fruits = ['apple', 'mandarin', 'orange', 'lemon']

X_fruits_2d = fruits[['height', 'width']]
y_fruits_2d = fruits['fruit_label']

# synthetic dataset for simple regression
from sklearn.datasets import make_regression
plt.figure()
plt.title('Sample regression problem with one input variable')
X_R1, y_R1 = make_regression(n_samples = 100, n_features=1,
                            n_informative=1, bias = 150.0,
                            noise = 30, random_state=0)
plt.scatter(X_R1, y_R1, marker= 'o', s=50)
plt.show()

# synthetic dataset for more complex regression
from sklearn.datasets import make_friedman1
plt.figure()
plt.title('Complex regression problem with one input variable')
X_F1, y_F1 = make_friedman1(n_samples = 100, n_features = 7,
                           random_state=0)

plt.scatter(X_F1[:, 2], y_F1, marker= 'o', s=50)
plt.show()

# synthetic dataset for classification (binary)
plt.figure()
plt.title('Sample binary classification problem with two informative features')
X_C2, y_C2 = make_classification(n_samples = 100, n_features=2,
                                n_redundant=0, n_informative=2,
                                n_clusters_per_class=1, flip_y = 0.1,
                                class_sep = 0.5, random_state=0)
plt.scatter(X_C2[:, 0], X_C2[:, 1], marker= 'o',
           c=y_C2, s=50, cmap=cmap_bold)
plt.show()

# more difficult synthetic dataset for classification (binary)
# with classes that are not linearly separable
```

```
X_D2, y_D2 = make_blobs(n_samples = 100, n_features = 2,
                        centers = 8, cluster_std = 1.3,
                        random_state = 4)
y_D2 = y_D2 % 2
plt.figure()
plt.title('Sample binary classification problem with non-linearly separable clas
ses')
plt.scatter(X_D2[:,0], X_D2[:,1], c=y_D2,
           marker= 'o', s=50, cmap=cmap_bold)
plt.show()

# Breast cancer dataset for classification
cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# Communities and Crime dataset
(X_crime, y_crime) = load_crime_dataset()
```


Sample regression problem with one input variable
250
200
150
100
50
0
−2
−1
0
1
2


Complex regression problem with one input variable
25
20
15
10
5
0.0
0.2
0.4
0.6
0.8
1.0

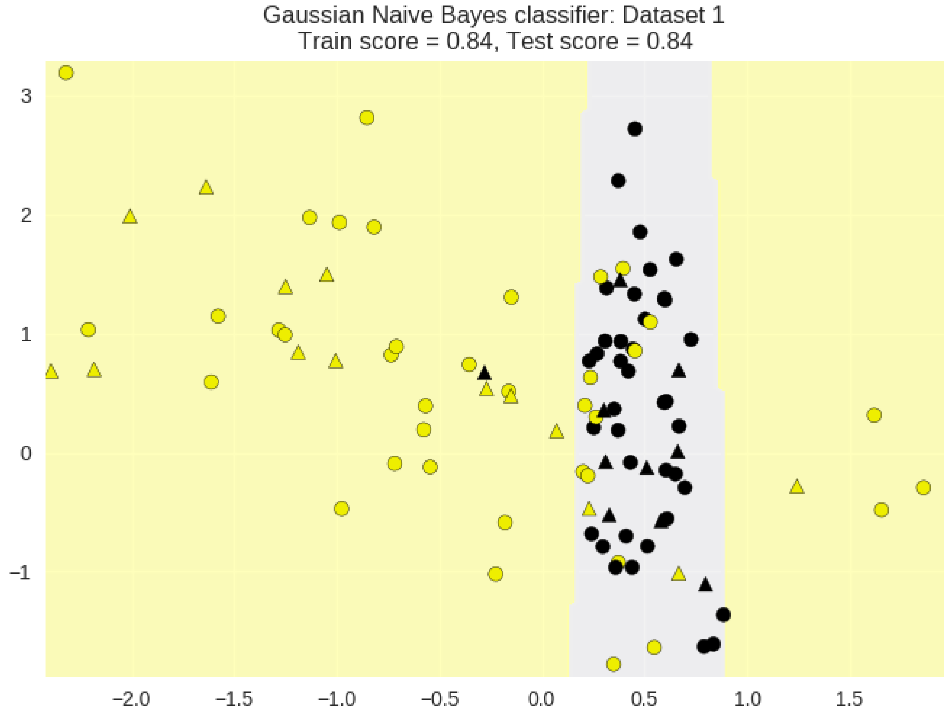

Sample binary classification problem with two informative features
3
2
1
0
−1
−2
−2
−1
0
1
2



04-01: Naive Bayes classifiers

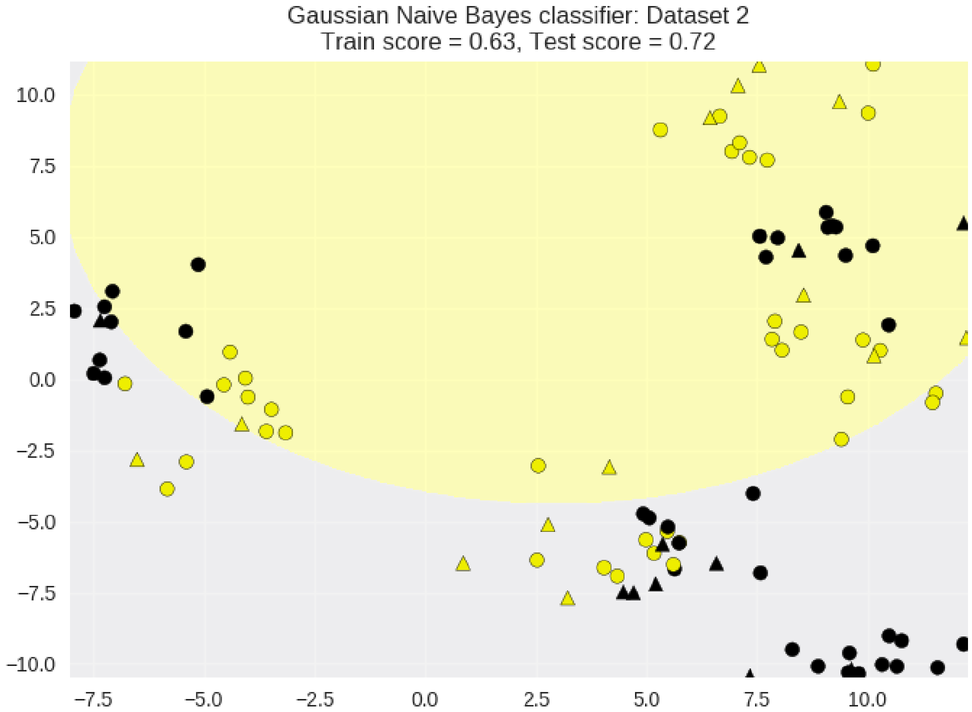
In [3]:

```
from sklearn.naive_bayes import GaussianNB # No additional models to control mod
el complexity.
from adspy_shared_utilities import plot_class_regions_for_classifier

X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2, random_state=0)
# Create the classifier and fit the parameters - do your usual routine
nbclf = GaussianNB().fit(X_train, y_train)
plot_class_regions_for_classifier(nbclf, X_train, y_train, X_test, y_test,
                                 'Gaussian Naive Bayes classifier: Dataset 1')
```



In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                    random_state=0)

nbclf = GaussianNB().fit(X_train, y_train)
plot_class_regions_for_classifier(nbclf, X_train, y_train, X_test, y_test,
                                 'Gaussian Naive Bayes classifier: Dataset 2')
```



Application to a real-world dataset

In [5]:

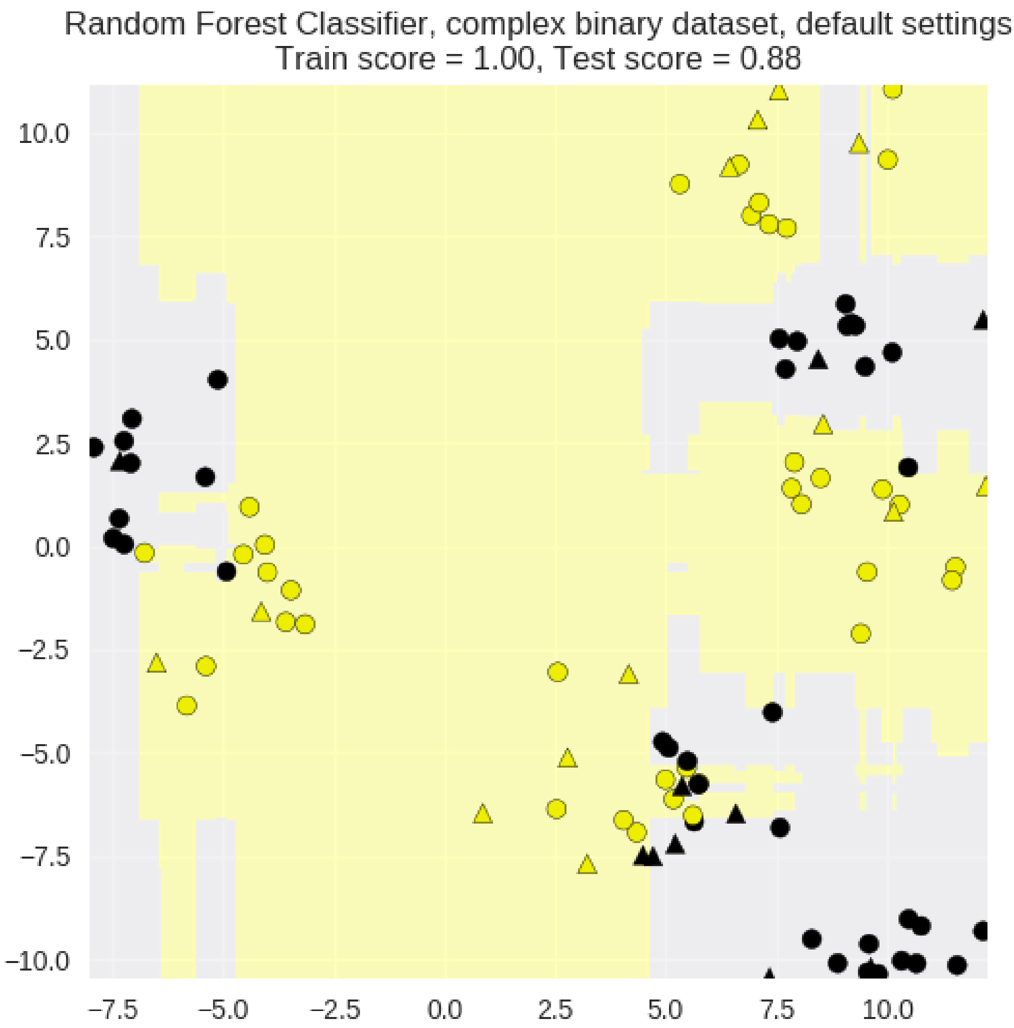
```
X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_s
tate = 0)

nbclf = GaussianNB().fit(X_train, y_train)
print('Breast cancer dataset')
print('Accuracy of GaussianNB classifier on training set: {:.2f}'
     .format(nbclf.score(X_train, y_train)))
print('Accuracy of GaussianNB classifier on test set: {:.2f}'
     .format(nbclf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of GaussianNB classifier on training set: 0.95
Accuracy of GaussianNB classifier on test set: 0.94
```

04-02: Ensembles of Decision Trees - Random Forests

In [6]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                    random_state = 0)
fig, subaxes = plt.subplots(1, 1, figsize=(6, 6))

clf = RandomForestClassifier().fit(X_train, y_train)
title = 'Random Forest Classifier, complex binary dataset, default settings'
plot_class_regions_for_classifier_subplot(clf, X_train, y_train, X_test,
                                         y_test, title, subaxes)

plt.show()
```

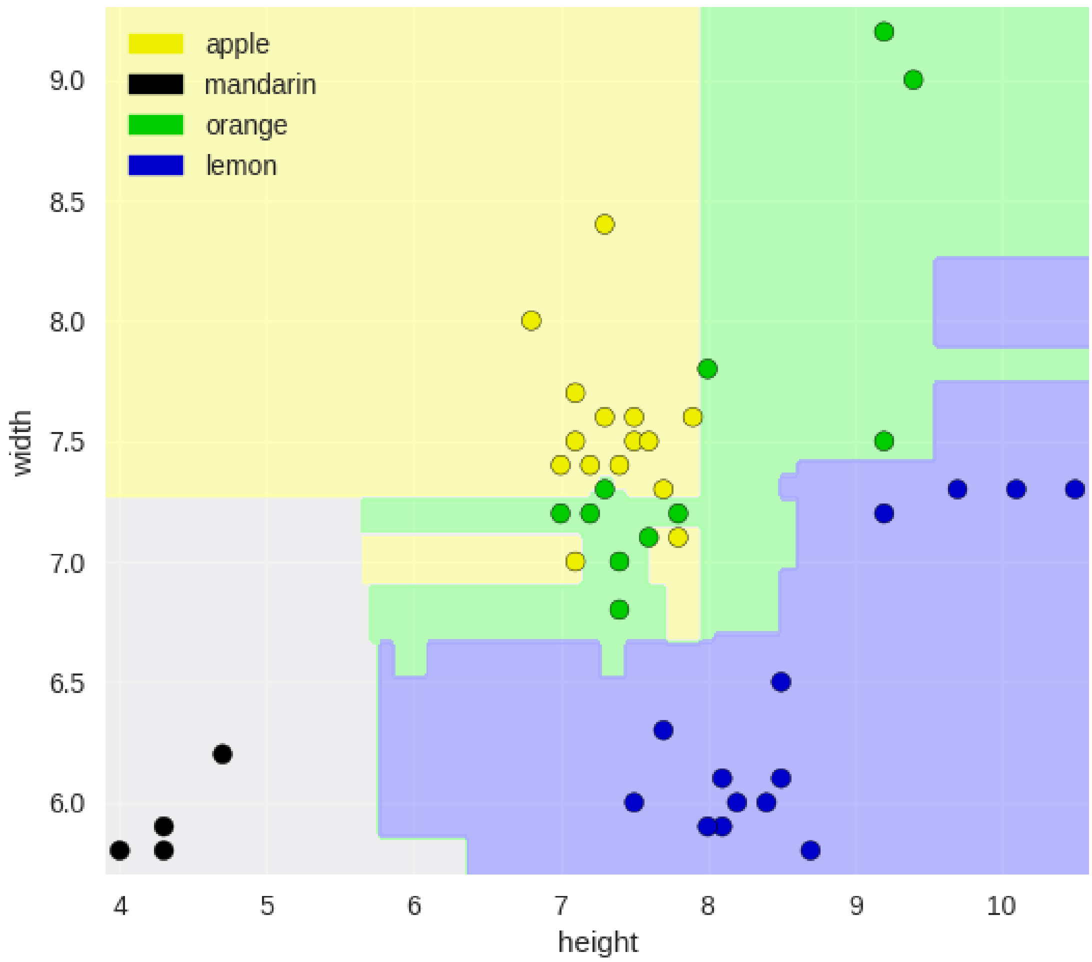


Random forest: Fruit dataset

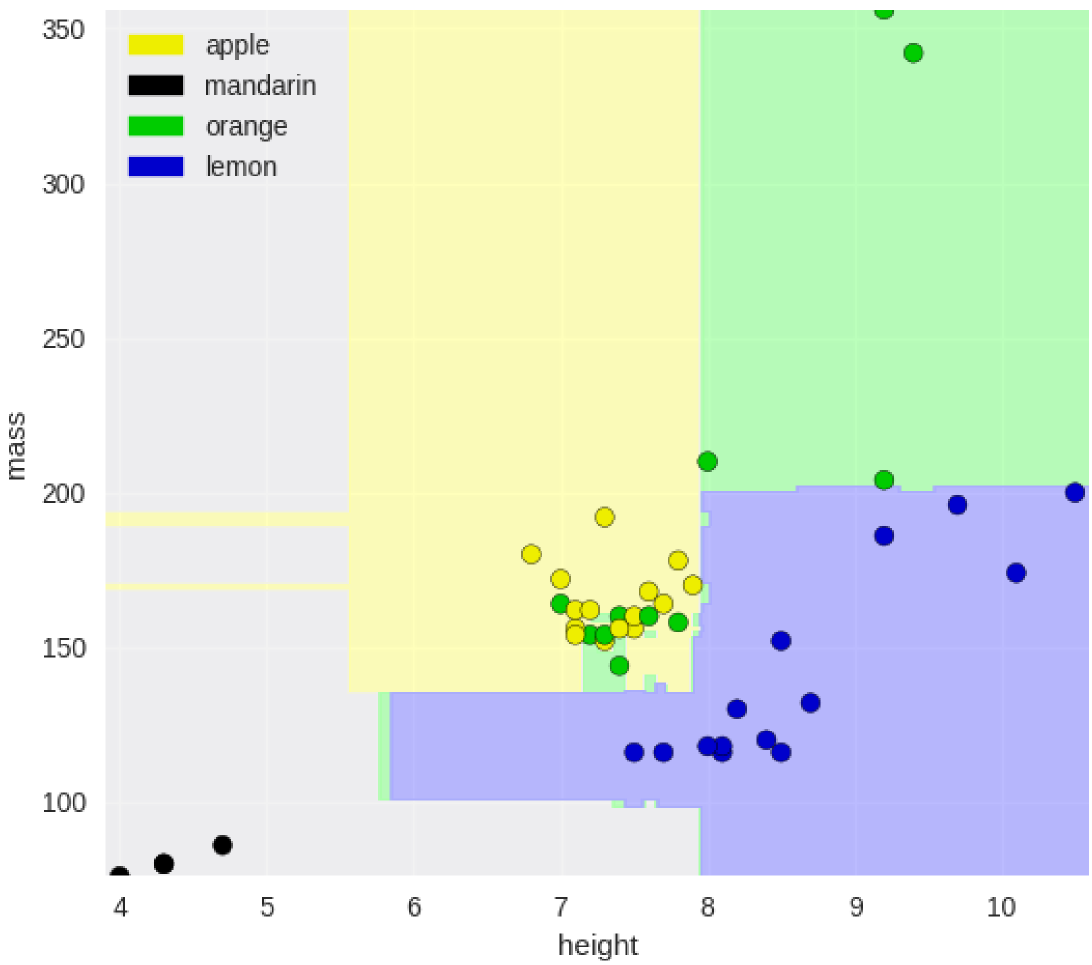
In [7]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot

X_train, X_test, y_train, y_test = train_test_split(X_fruits.as_matrix(),
                                                    y_fruits.as_matrix(),
                                                    random_state = 0)
fig, subaxes = plt.subplots(6, 1, figsize=(6, 32))

title = 'Random Forest, fruits dataset, default settings'
# After creating the train-test split on the data, we iterate through each pair
 of feature columns
# on the training set.
pair_list = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]
#####  -------- To plot the Random Forest on the fruit dataset --------------
for pair, axis in zip(pair_list, subaxes):
    X = X_train[:, pair]
    y = y_train
    # Create the random forest classifier and fit the model
    # For each pair of features we call the fit method on that subset of the tra
ining data X
    # using the labels y
    clf = RandomForestClassifier().fit(X, y)
    # Use the utility function to plot this graph
    plot_class_regions_for_classifier_subplot(clf, X, y, None,
                                             None, title, axis,
                                             target_names_fruits)

    axis.set_xlabel(feature_names_fruits[pair[0]])
    axis.set_ylabel(feature_names_fruits[pair[1]])

plt.tight_layout()
plt.show()
### ---------------- To see the score of the classifier on default parameters --
---------
clf = RandomForestClassifier(n_estimators = 10,
                            random_state=0).fit(X_train, y_train)

print('Random Forest, Fruit dataset, default settings')
print('Accuracy of RF classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of RF classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

Random Forest, fruits dataset, default settings

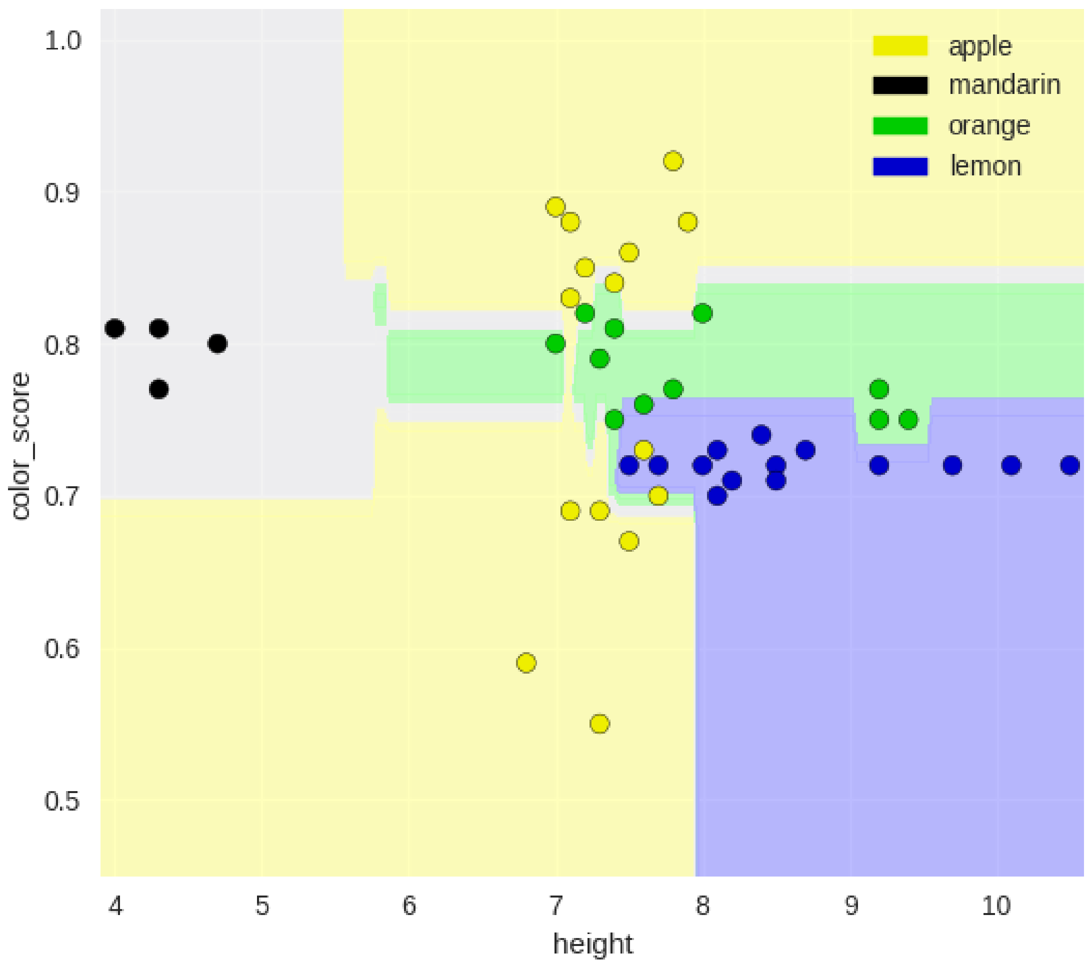


Random Forest, fruits dataset, default settings

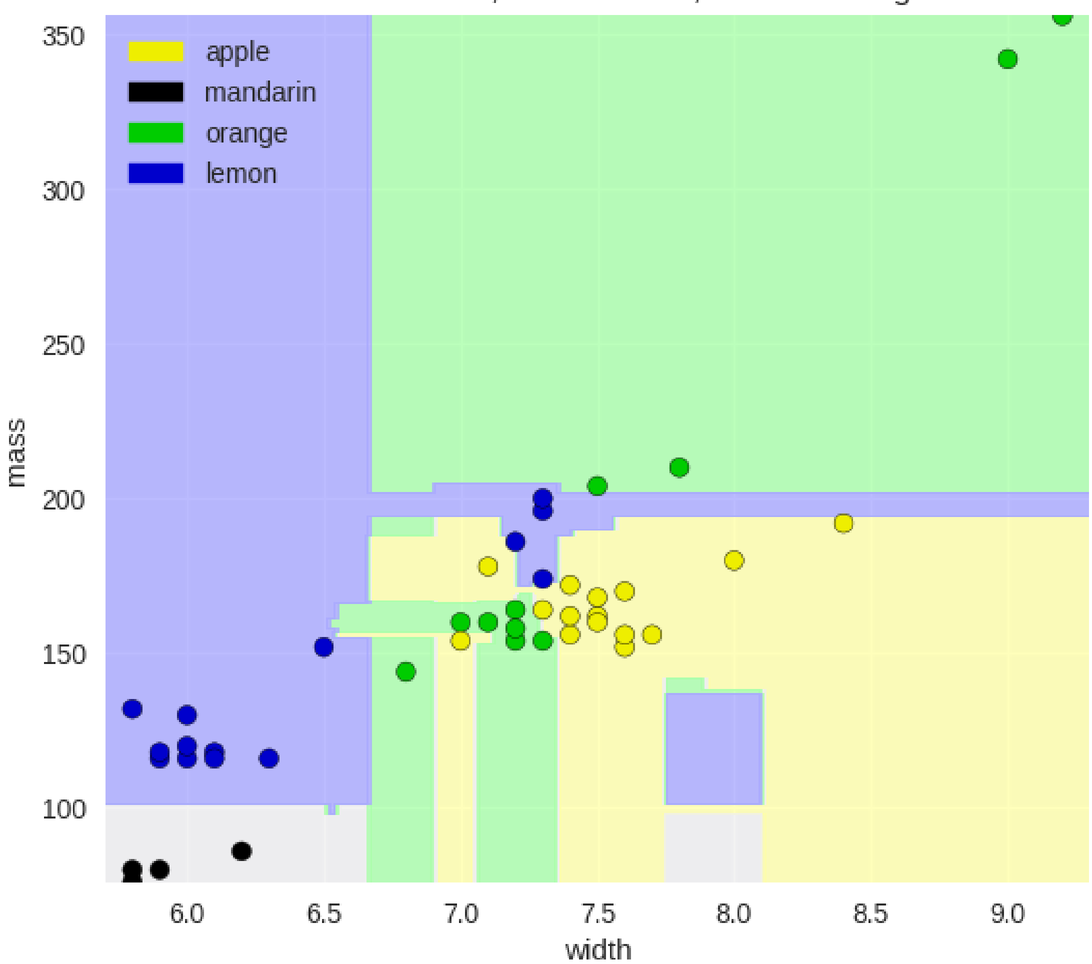


Random Forest, fruits dataset, default settings

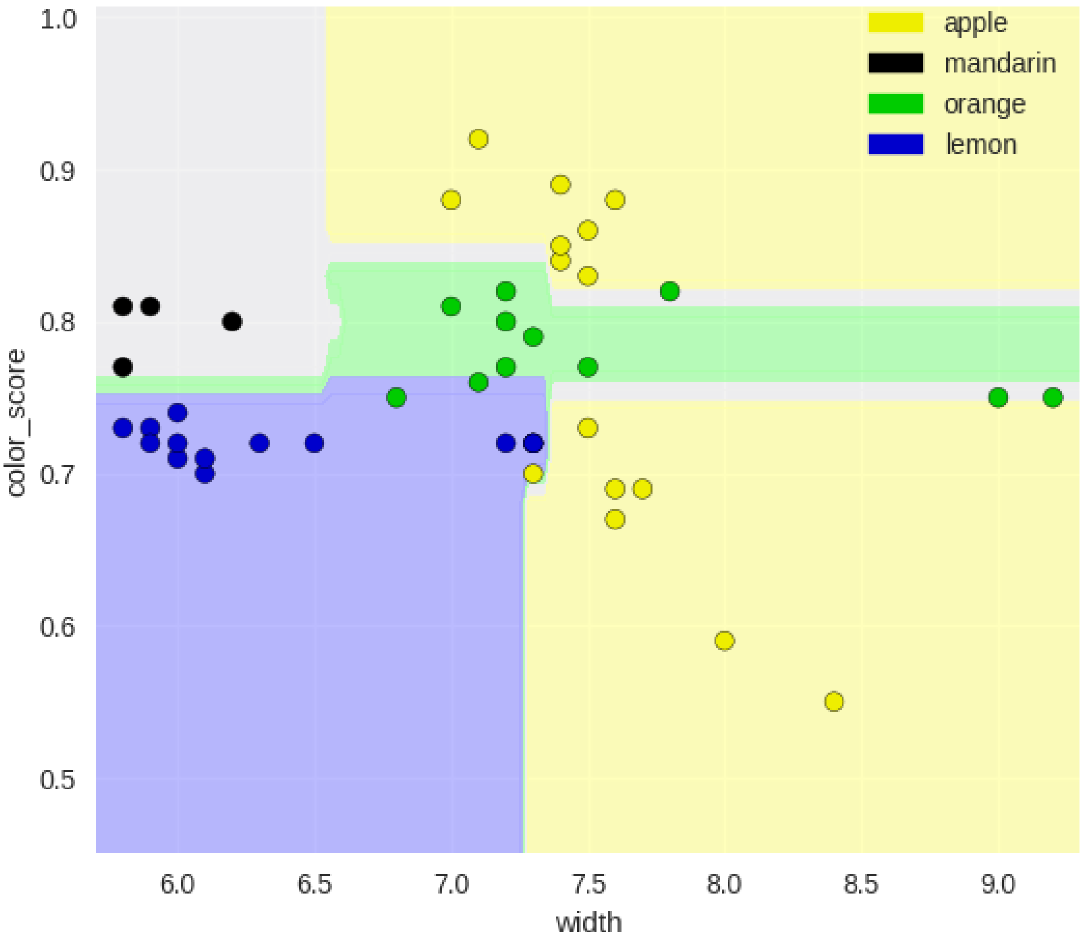
Random Forest, fruits dataset, default settings



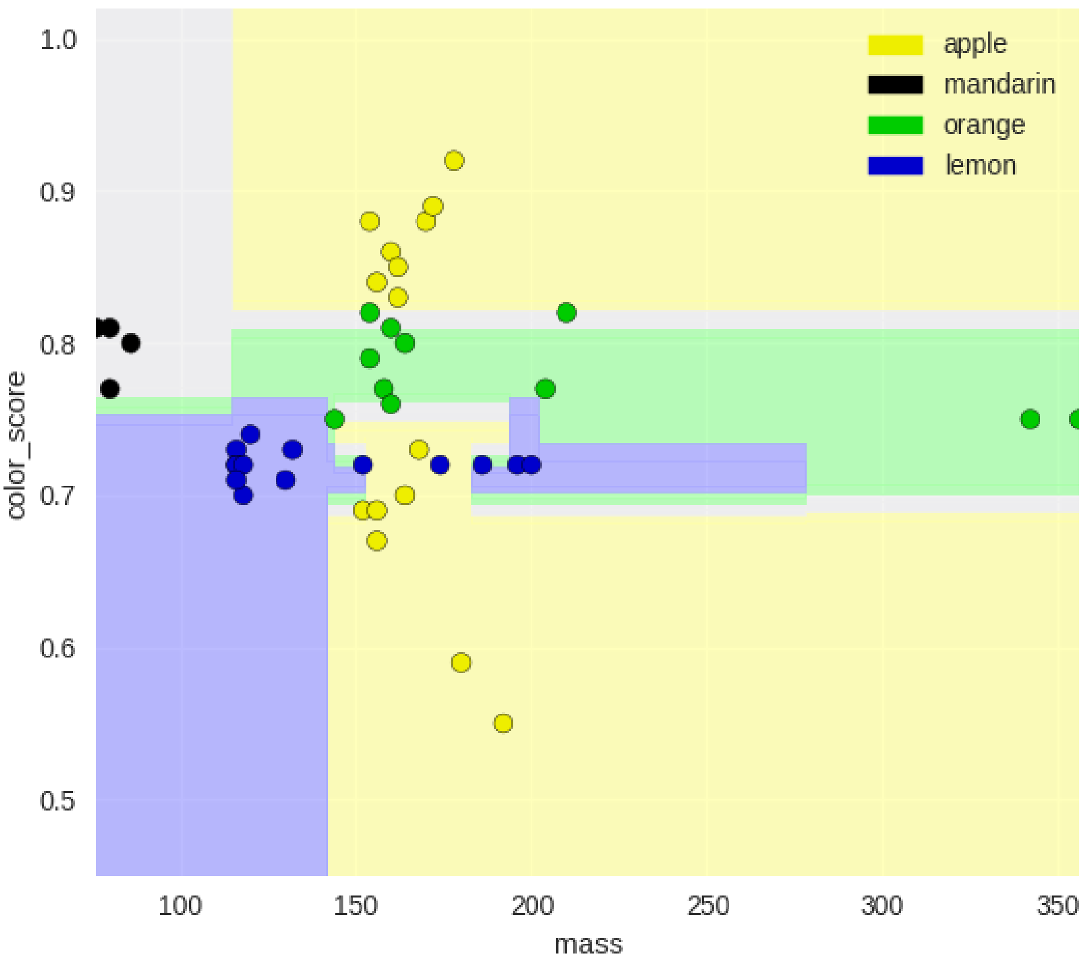
Random Forest, fruits dataset, default settings



Random Forest, fruits dataset, default settings



Random Forest, fruits dataset, default settings



```
Random Forest, Fruit dataset, default settings
Accuracy of RF classifier on training set: 1.00
Accuracy of RF classifier on test set: 0.80
```

Random Forests on a real-world dataset

In [8]:

```
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_s
tate = 0)
# We apply random forest on the breast cancer dataset - and set the max-features
here to 8.
# Note the RandomForestClassifier also requires a random_state variable.
clf = RandomForestClassifier(max_features = 8, random_state = 0)
clf.fit(X_train, y_train)

print('Breast cancer dataset')
print('Accuracy of RF classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of RF classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of RF classifier on training set: 1.00
Accuracy of RF classifier on test set: 0.99
```

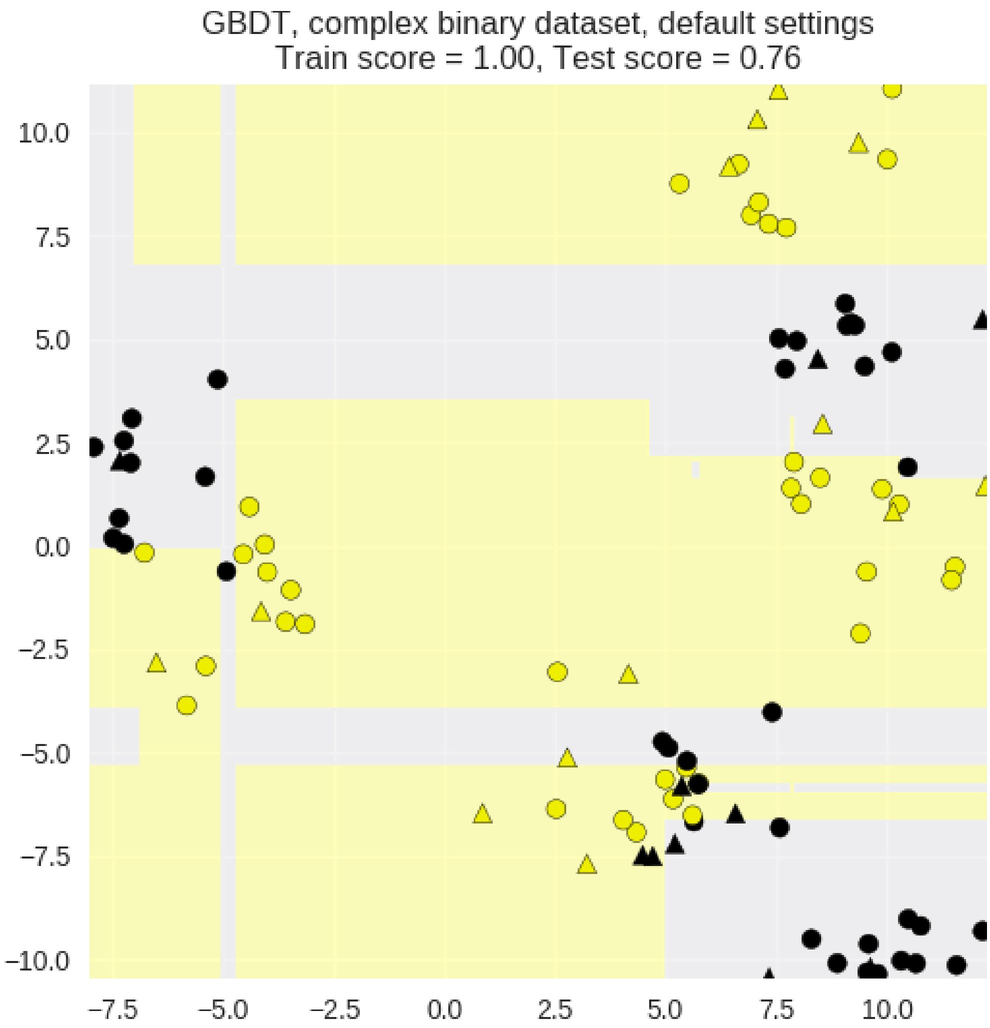
We see that this random forest - **with no feature scaling or parameter tuning** - achieves very good test set performance on this dataset. In fact, it is as good or better than all the other supervised methods we've seen so far including kernelised SVMs and neural networks.

Note that we **did not** have to do scaling or any preprocessing of the data. This is an advantage of using random forests. Also note that we passed in a `random_state = 0` parameter in order to make the results reproducible. If we didn't set the `random_state` parameter, the model will likely be different each time due to the randomised nature of the random forest algorithm.

04-03: Ensemble Trees: Gradient-boosted decision trees

In [9]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state = 0
)
fig, subaxes = plt.subplots(1, 1, figsize=(6, 6))
# Create the Gradient Boosting Classifier object
clf = GradientBoostingClassifier().fit(X_train, y_train)
title = 'GBDT, complex binary dataset, default settings'
plot_class_regions_for_classifier_subplot(clf, X_train, y_train, X_test,
                                          y_test, title, subaxes)

plt.show()
```

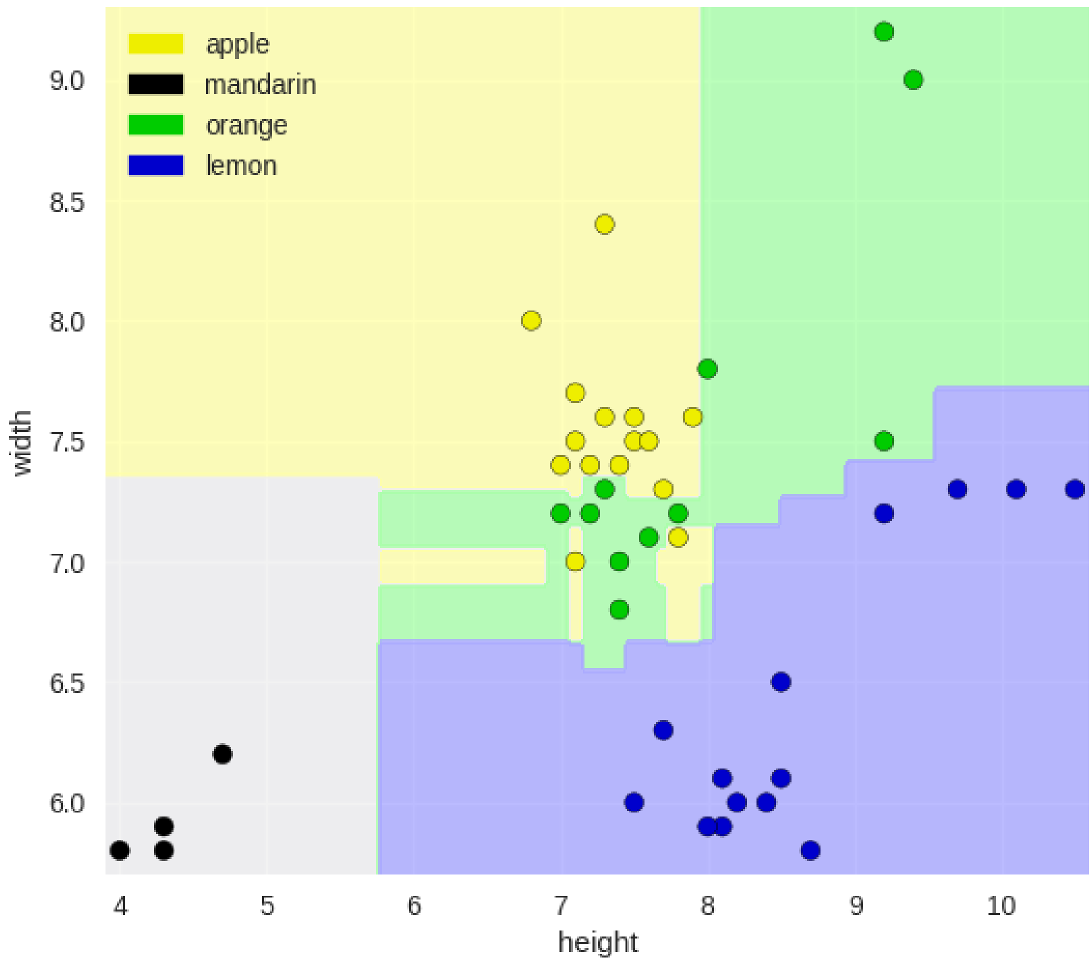


Gradient boosted decision trees on the fruit dataset

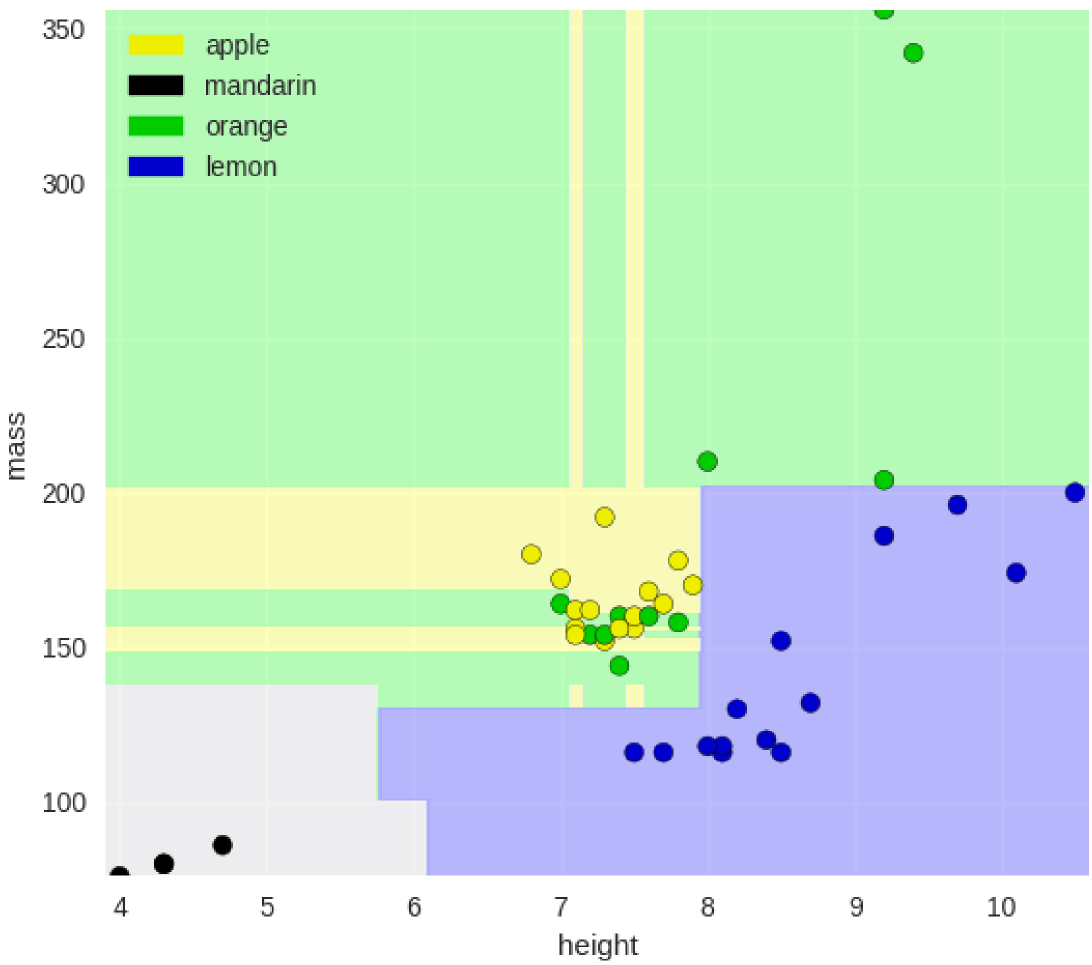
In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(X_fruits.as_matrix(),
                                                   y_fruits.as_matrix(),
                                                   random_state = 0)
fig, subaxes = plt.subplots(6, 1, figsize=(6, 32))

pair_list = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]

for pair, axis in zip(pair_list, subaxes):
    X = X_train[:, pair]
    y = y_train

    clf = GradientBoostingClassifier().fit(X, y)
    plot_class_regions_for_classifier_subplot(clf, X, y, None,
                                             None, title, axis,
                                             target_names_fruits)

    axis.set_xlabel(feature_names_fruits[pair[0]])
    axis.set_ylabel(feature_names_fruits[pair[1]])

plt.tight_layout()
plt.show()
clf = GradientBoostingClassifier().fit(X_train, y_train)

print('GBDT, Fruit dataset, default settings')
print('Accuracy of GBDT classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

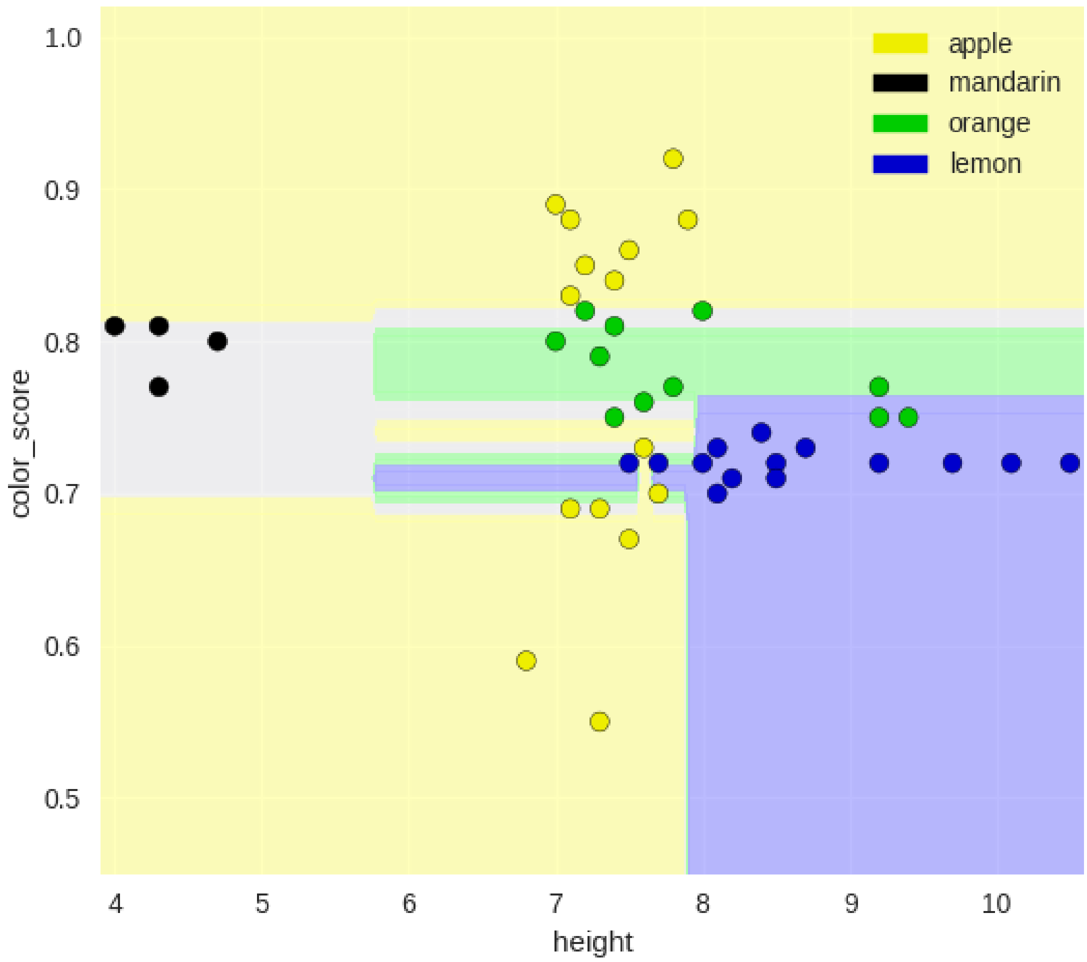
GBDT, complex binary dataset, default settings



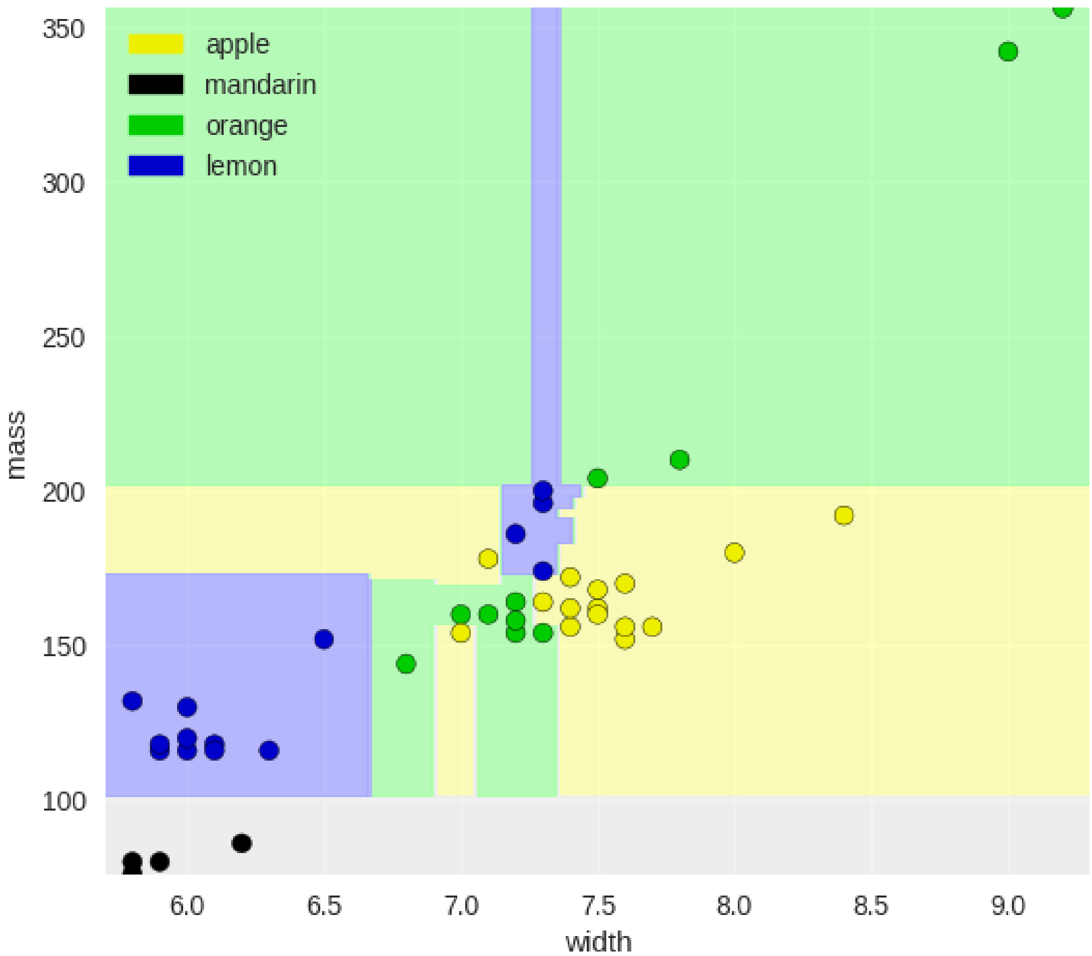
GBDT, complex binary dataset, default settings



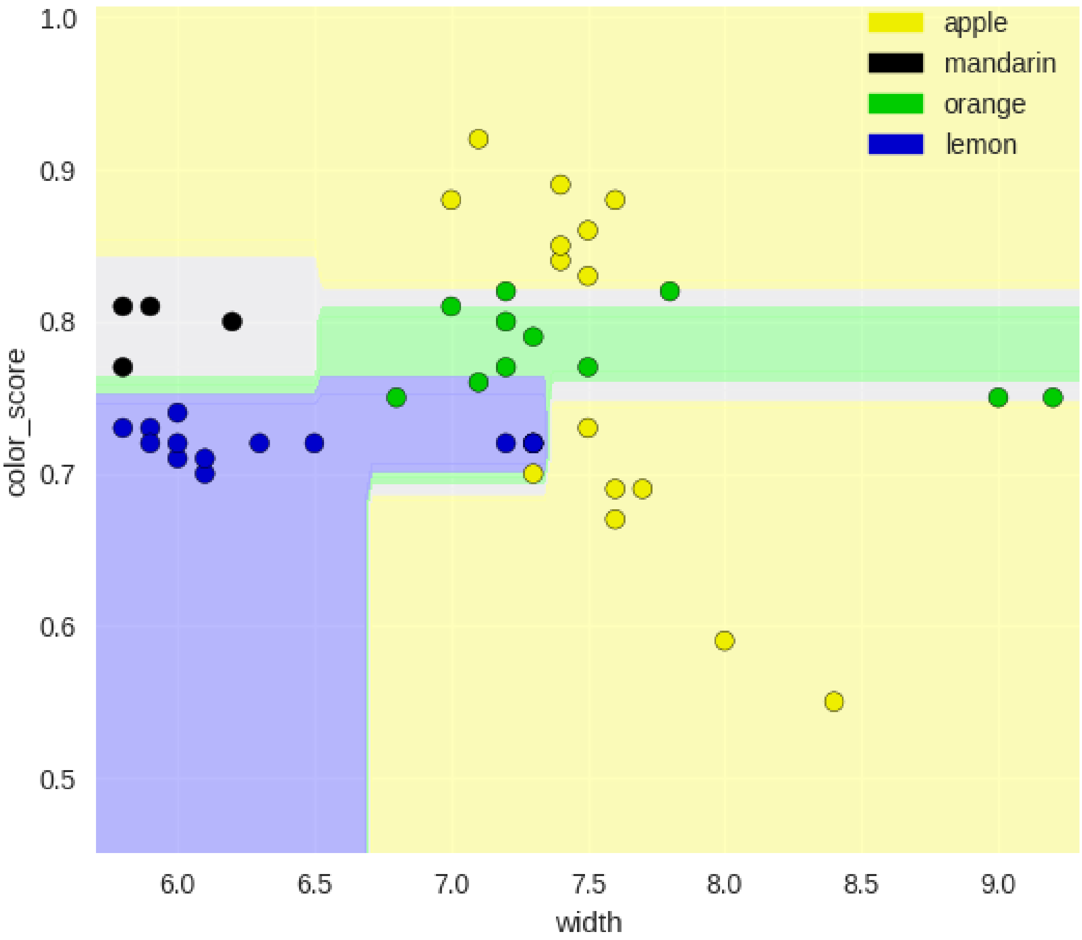
GBDT, complex binary dataset, default settings



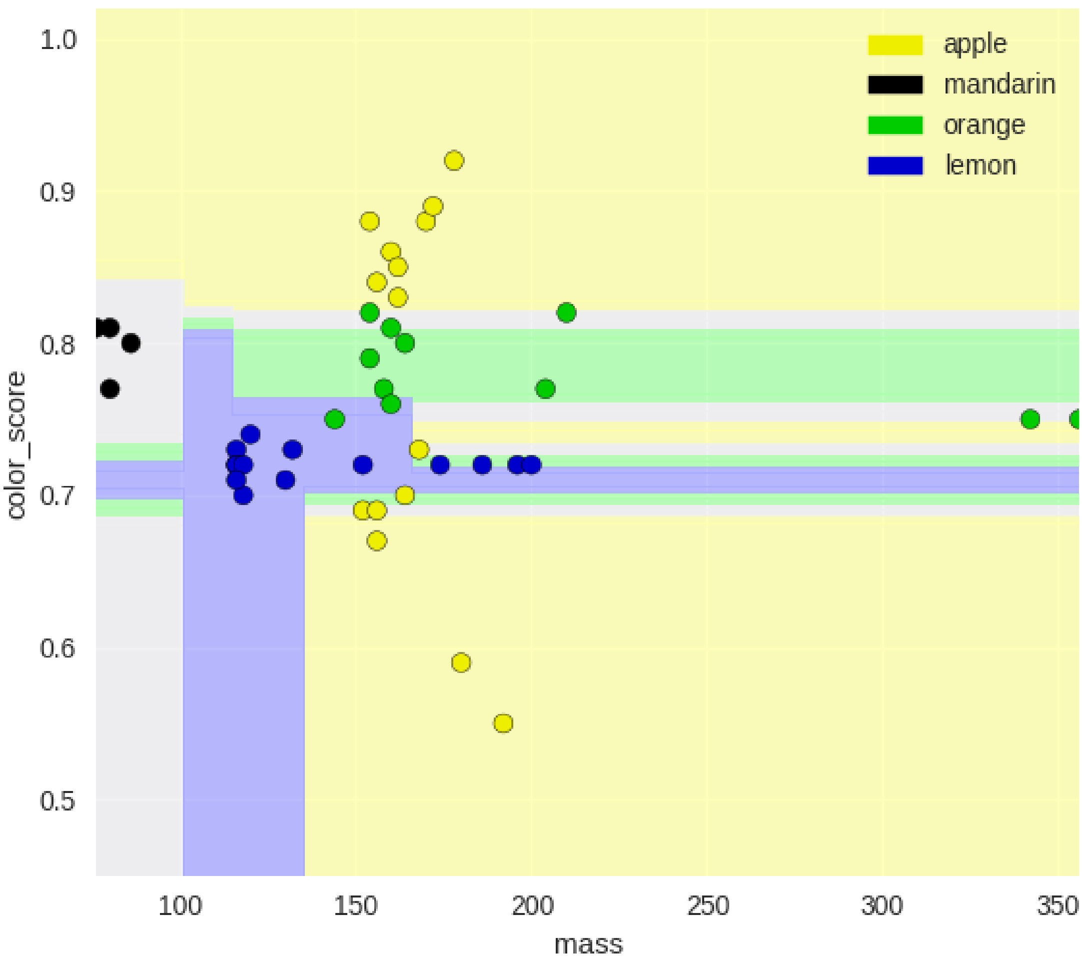
GBDT, complex binary dataset, default settings



GBDT, complex binary dataset, default settings



GBDT, complex binary dataset, default settings



```
GBDT, Fruit dataset, default settings
Accuracy of GBDT classifier on training set: 1.00
Accuracy of GBDT classifier on test set: 0.80
```

Gradient-boosted decision trees on a real-world dataset

In [11]:

```
from sklearn.ensemble import GradientBoostingClassifier

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_s
tate = 0)

clf = GradientBoostingClassifier(random_state = 0)
clf.fit(X_train, y_train)

print('Breast cancer dataset (learning_rate=0.1, max_depth=3)')
print('Accuracy of GBDT classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}\n'
     .format(clf.score(X_test, y_test)))

clf = GradientBoostingClassifier(learning_rate = 0.01, max_depth = 2, random_sta
te = 0)
clf.fit(X_train, y_train)

print('Breast cancer dataset (learning_rate=0.01, max_depth=2)')
print('Accuracy of GBDT classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset (learning_rate=0.1, max_depth=3)
Accuracy of GBDT classifier on training set: 1.00
Accuracy of GBDT classifier on test set: 0.96

Breast cancer dataset (learning_rate=0.01, max_depth=2)
Accuracy of GBDT classifier on training set: 0.97
Accuracy of GBDT classifier on test set: 0.97
```

Notice that the first run on the Breast cancer dataset has an accuracy of 1 on the training set - which implies that the model is overfitting.

- So we go about this by **reducing the `learning_rate`**, such that a **less** complex model is produced, and hence less likely to fix the mistakes of its predecessor.
- In addition, we can reduce the **`max_depth`** parameter for individual trees in the ensemble.

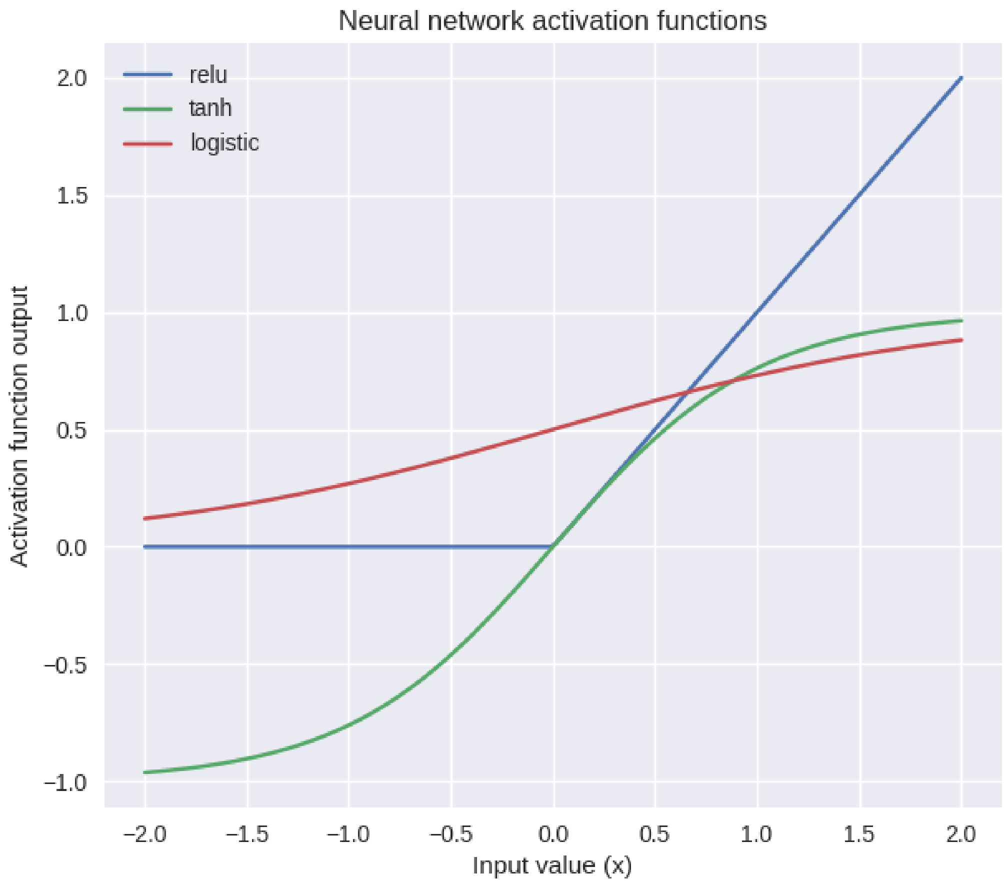
The 2nd classifier example makes these changes in the parameters, and you can see that the training set accuracy decreases slightly, with an increase in the test set accuracy.

04-04: Neural networks

Activation functions

In [12]:

```
xrange = np.linspace(-2, 2, 200)

plt.figure(figsize=(7,6))

plt.plot(xrange, np.maximum(xrange, 0), label = 'relu')
plt.plot(xrange, np.tanh(xrange), label = 'tanh')
plt.plot(xrange, 1 / (1 + np.exp(-xrange)), label = 'logistic')
plt.legend()
plt.title('Neural network activation functions')
plt.xlabel('Input value (x)')
plt.ylabel('Activation function output')

plt.show()
```

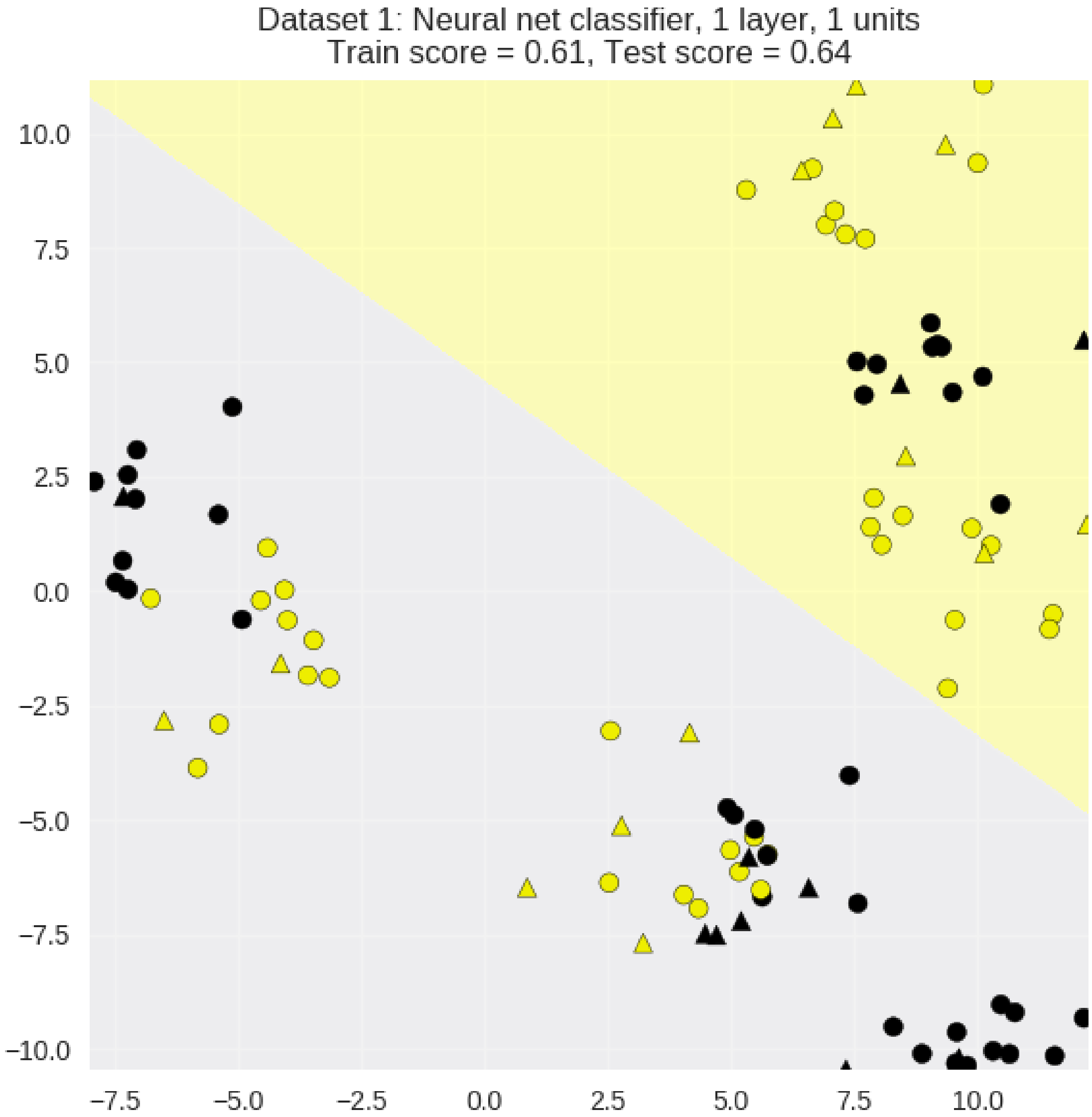


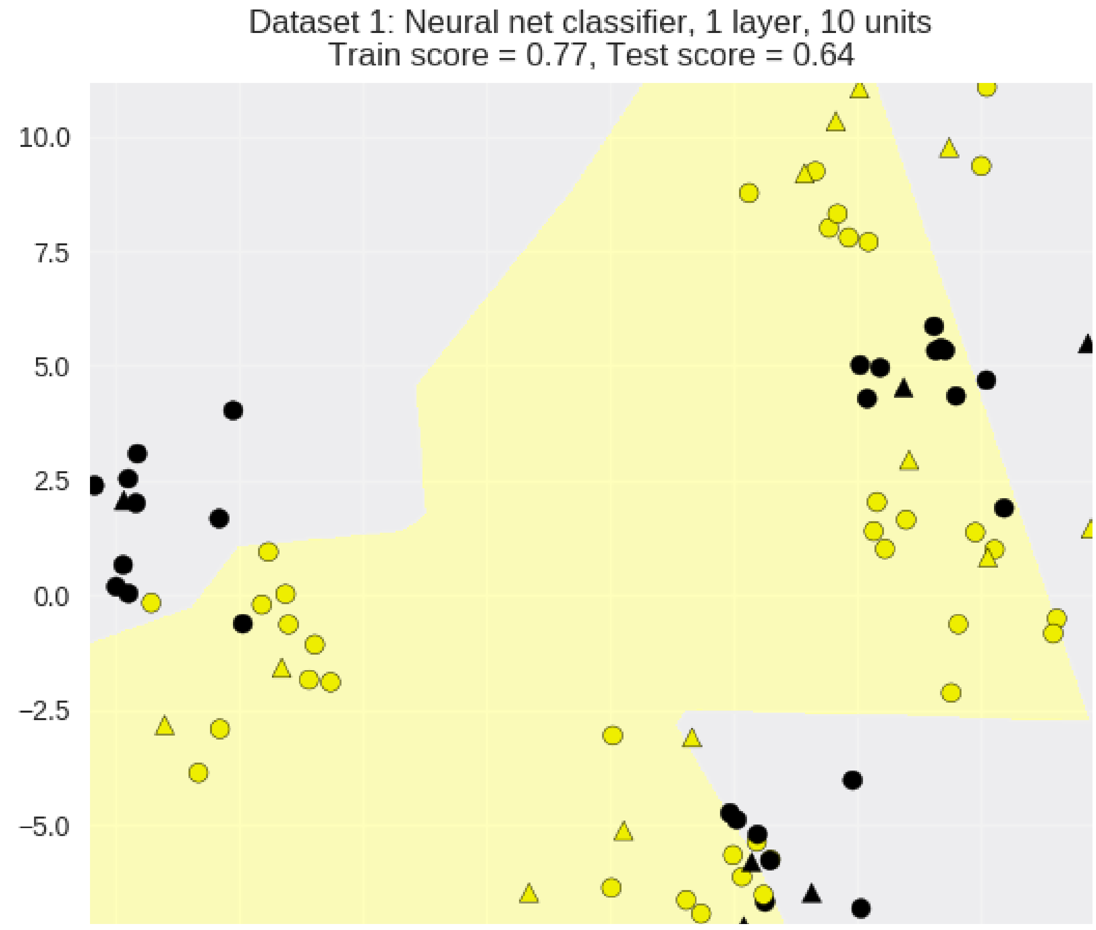
Neural networks: Classification

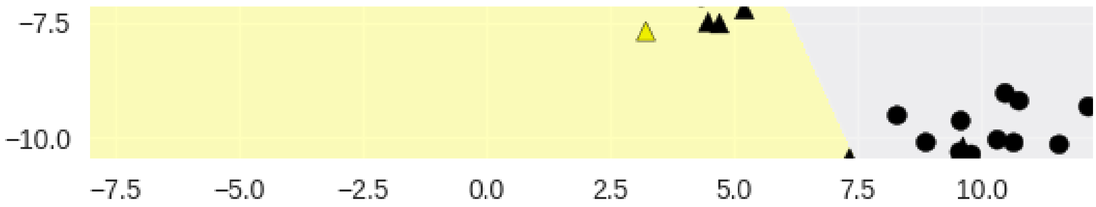
Synthetic dataset 1: single hidden layer

In [13]:

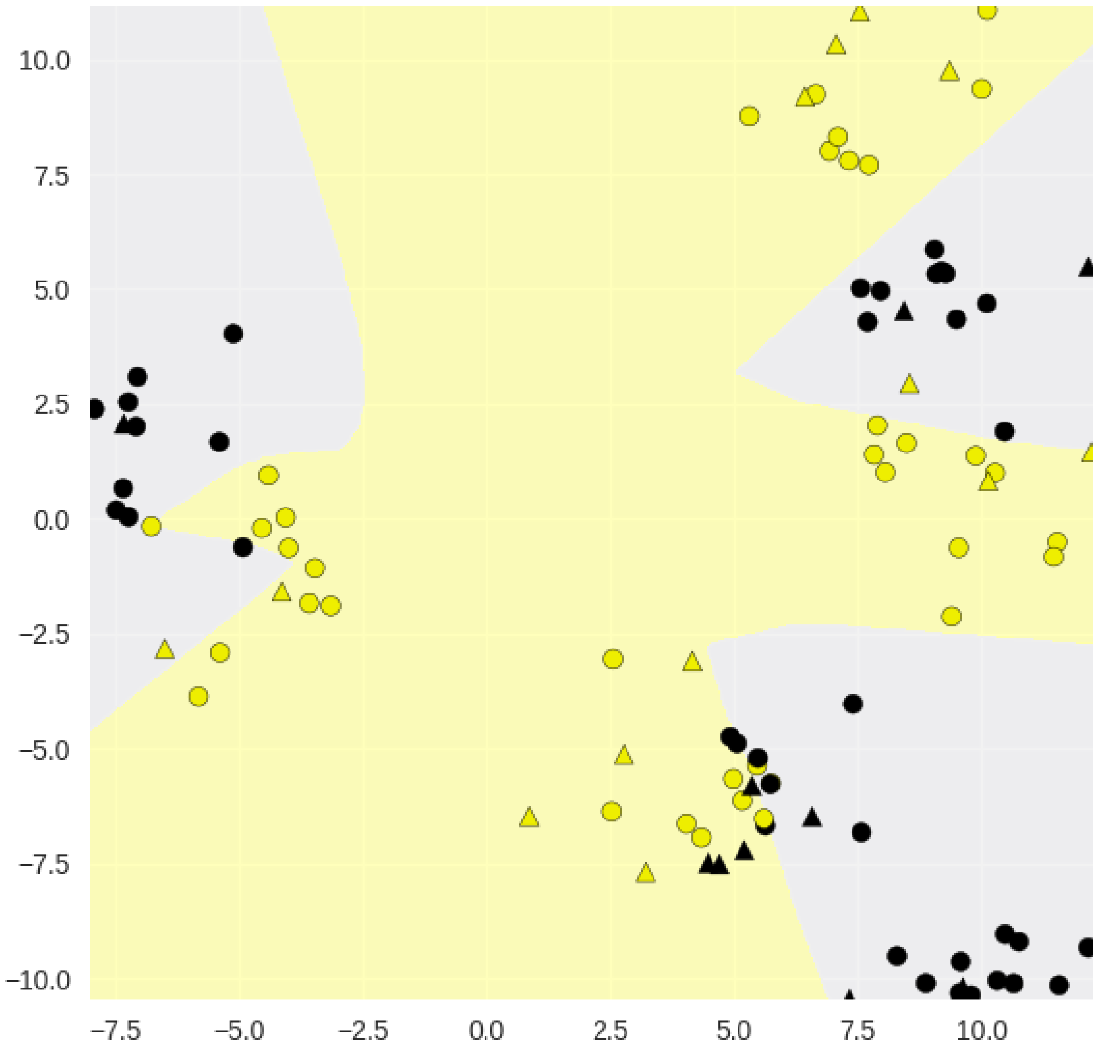
```
from sklearn.neural_network import MLPClassifier
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state=0)

fig, subaxes = plt.subplots(3, 1, figsize=(6,18))

for units, axis in zip([1, 10, 100], subaxes):
    # Create the classifier
    # note the parameter hidden_layer_sizes -> this parameter is a list, with 1
 element for each hidden layer
    # that gives the number of hidden units to use in this layer.
    # In this example, we would like 1 hidden layer, using the number in the var
iable called "units"
    # By default, if you don't specify the hidden_layer_sizes parameter,scikit l
earn will create a single hidden layer
    # with 100 hidden units.
    #
    # While a setting of 10 may work well like the one we use as examples here,
 for really complex data sets,
    # the number of hidden units could be in the thousands.
    # It's also possible to create an MLP ( in a later example) with more than 1
hidden layer, by passing in the
    # hidden_layer_sizes parameter with multiple entries.
    #
    # Note the use of the extra parameter here, called "Solver". This specifies
 the algorithm to use for learning
    # the weights of the network. Here we use the LBFGS algorithm, we'll discuss
the solver parameter setting further,
    # at the end of the lecture.
    #
    # Also note that we are passing in a random state parameter when creating th
e MLP Classifier object, like we did
    # for the train-test split function, and we set its value to 0.
    #
    # This is because for neural networks, their weights are initialised randoml
y, which can affect the model that
    # is learned. Because of this, even without changing the key parameters on t
he dataset, the same neural network
    # algorithm might learn two different models, depending on the value of the
 internal random seed chosen. By
    # choosing the same random seed, we can then guarantee reproducible results.


    # OKAY, so create the classifier with appropriate params, and fit the model
 with the training data.
    nnclf = MLPClassifier(hidden_layer_sizes = [units], solver='lbfgs',
                         random_state = 0).fit(X_train, y_train)

    title = 'Dataset 1: Neural net classifier, 1 layer, {} units'.format(units)
    # MPL Support
    plot_class_regions_for_classifier_subplot(nnclf, X_train, y_train,
                                             X_test, y_test, title, axis)
    plt.tight_layout()
```


Dataset 1: Neural net classifier, 1 layer, 1 units
Train score = 0.61, Test score = 0.64
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Dataset 1: Neural net classifier, 1 layer, 10 units
Train score = 0.77, Test score = 0.64
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0



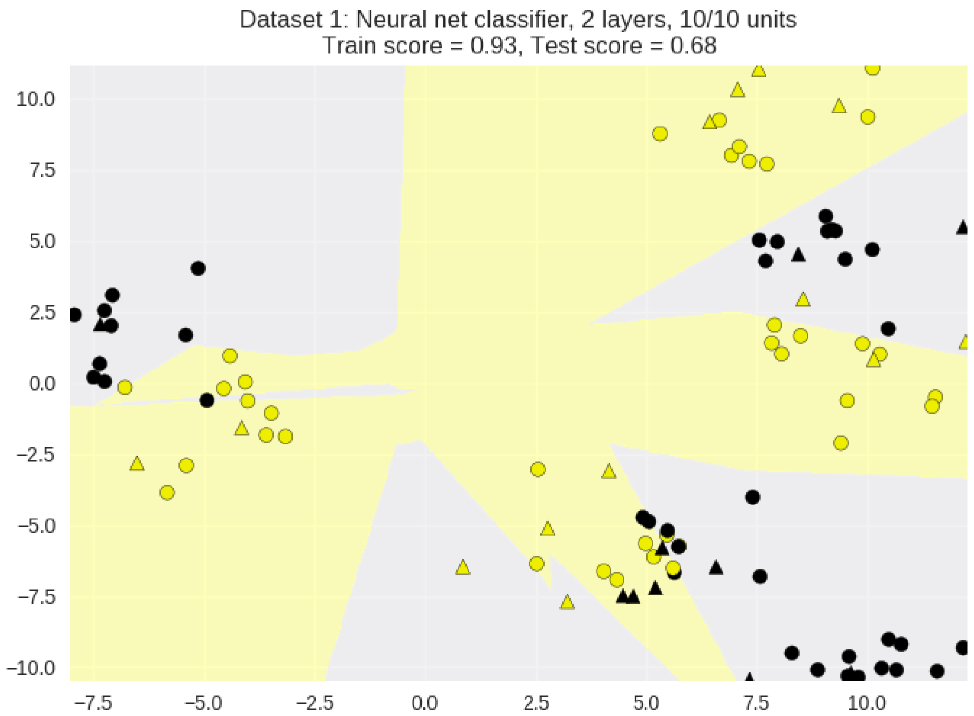
Dataset 1: Neural net classifier, 1 layer, 100 units
Train score = 0.93, Test score = 0.76



Synthetic dataset 1: two hidden layers

In [14]:

```
from adspy_shared_utilities import plot_class_regions_for_classifier

X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state=0)

# This is how you create the 2 hidden layers - change the hidden layer sizes par
ameter to a 2 element list.
nnclf = MLPClassifier(hidden_layer_sizes = [10, 10], solver='lbfgs',
                     random_state = 0).fit(X_train, y_train)

plot_class_regions_for_classifier(nnclf, X_train, y_train, X_test, y_test,
                                 'Dataset 1: Neural net classifier, 2 layers, 1
0/10 units')
```

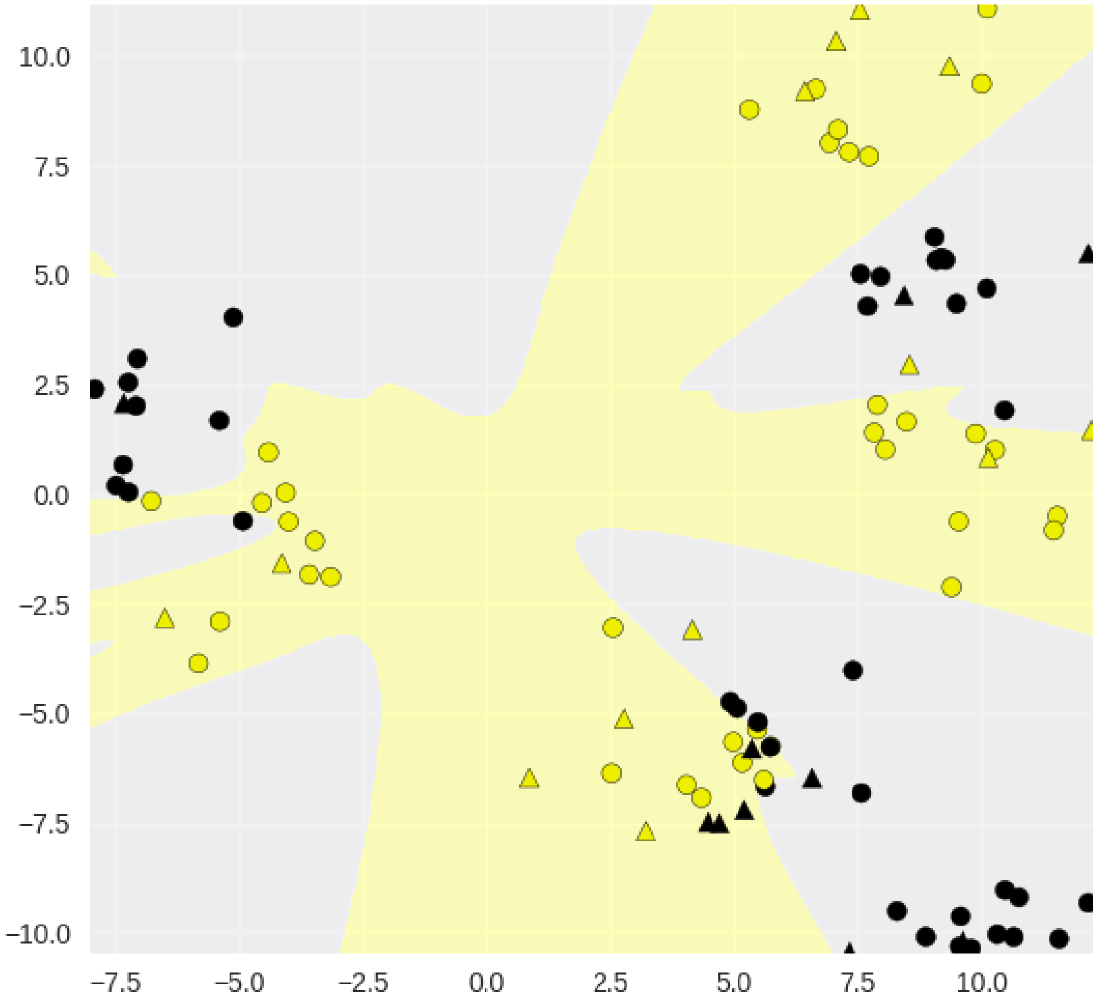


Regularization parameter: alpha

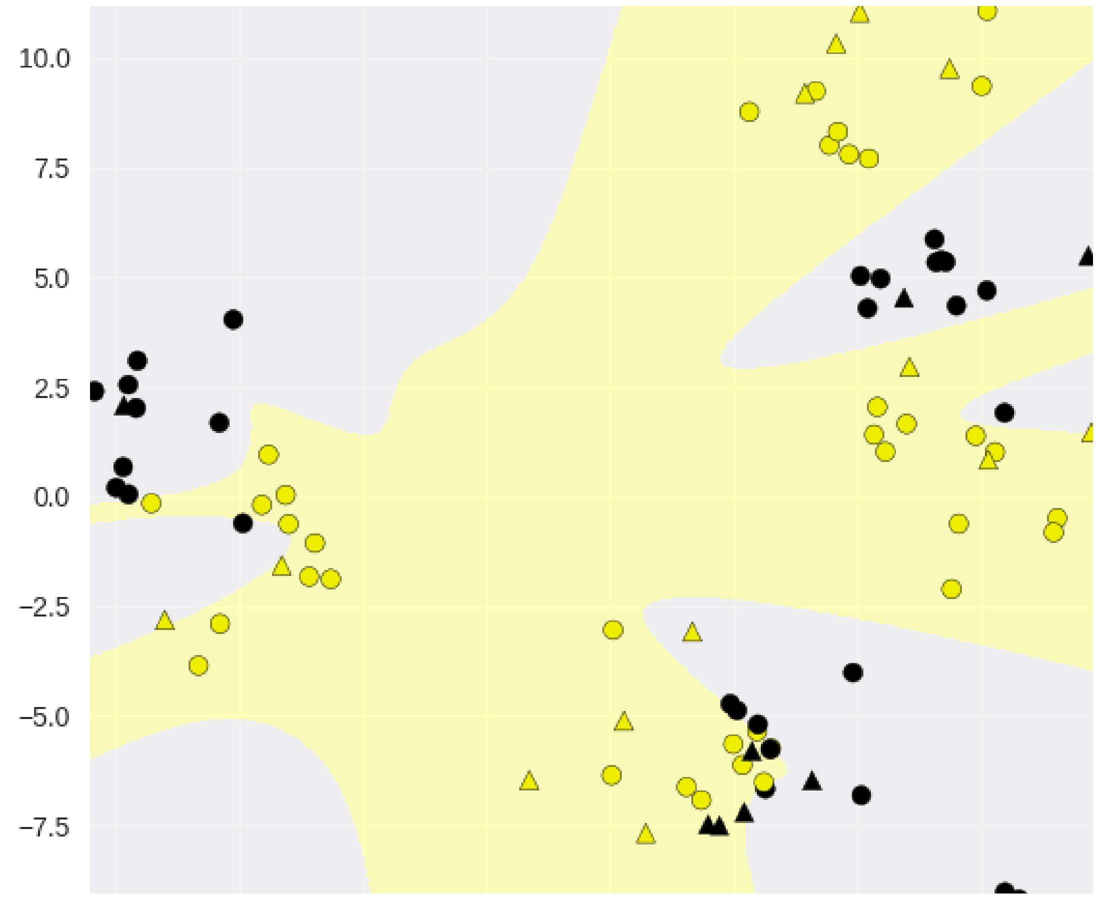
In [15]:

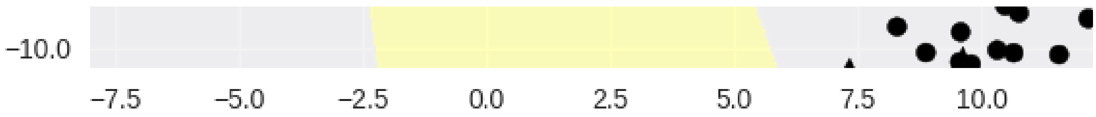
```
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state=0)

fig, subaxes = plt.subplots(4, 1, figsize=(6, 23))

# This code shows an example of the effect of changing alpha on the neural netwo
rk.
for this_alpha, axis in zip([0.01, 0.1, 1.0, 5.0], subaxes):
    nnclf = MLPClassifier(solver='lbfgs', activation = 'tanh', # For variety we
 set the activation function to the
                                                                # hyperbolic tan
gent function
                         alpha = this_alpha, # by default this is set to 0.0001
                         hidden_layer_sizes = [100, 100],
                         random_state = 0).fit(X_train, y_train)

    title = 'Dataset 2: NN classifier, alpha = {:.3f} '.format(this_alpha)

    plot_class_regions_for_classifier_subplot(nnclf, X_train, y_train,
                                             X_test, y_test, title, axis)
    plt.tight_layout()
```

Dataset 2: NN classifier, alpha = 0.010
Train score = 0.97, Test score = 0.72

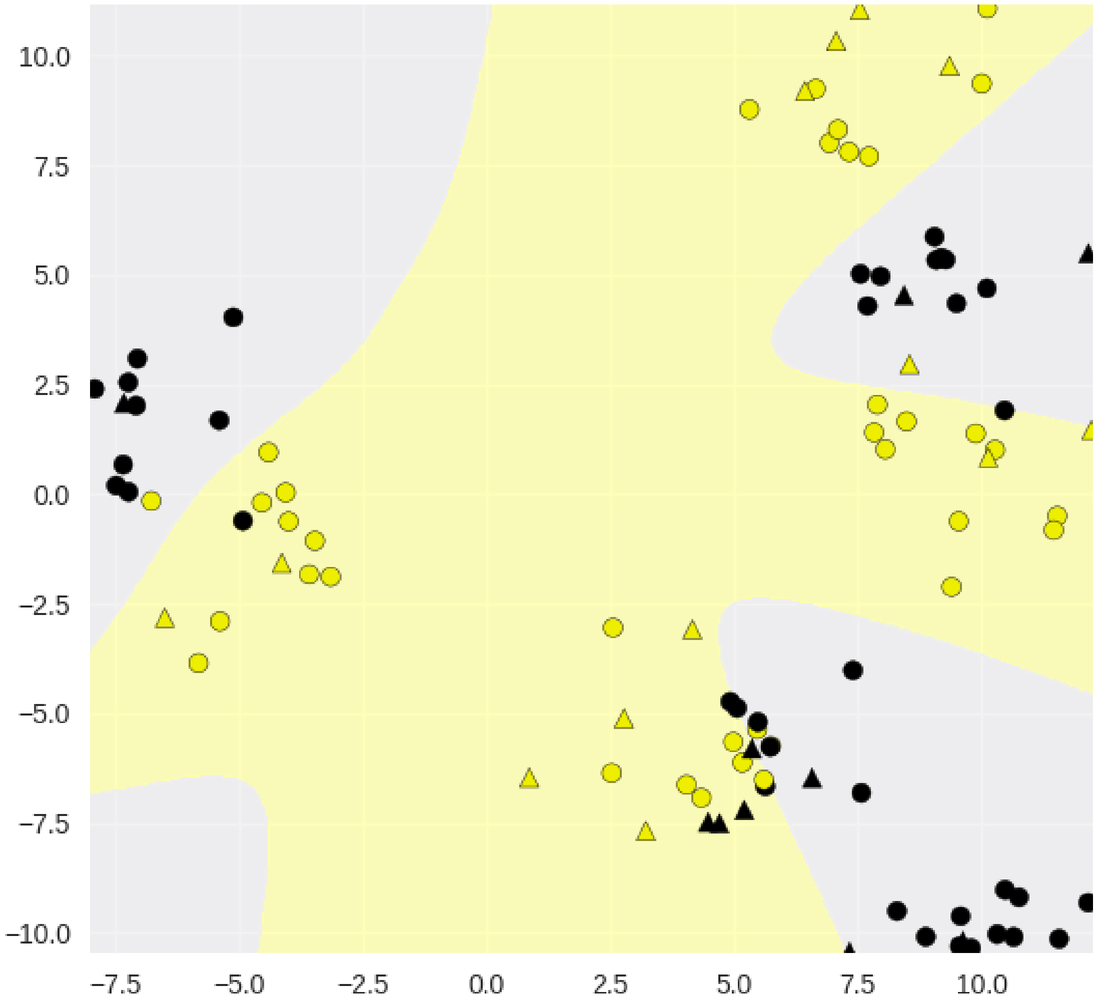


Dataset 2: NN classifier, alpha = 0.100
Train score = 0.97, Test score = 0.72

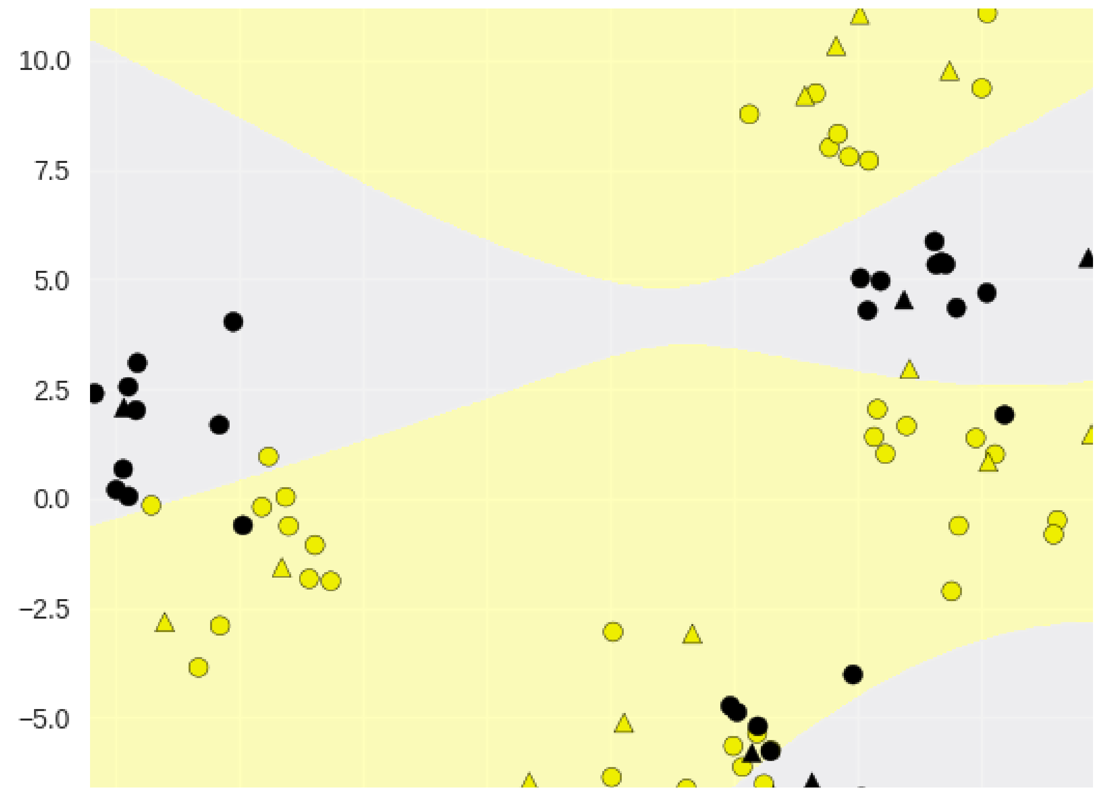


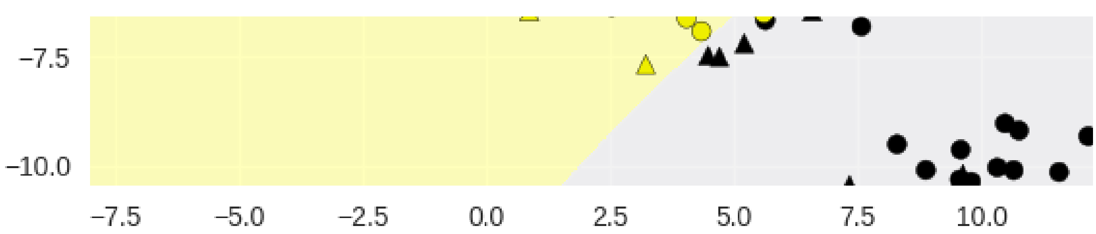


Dataset 2: NN classifier, alpha = 1.000
Train score = 0.89, Test score = 0.80



Dataset 2: NN classifier, alpha = 5.000
Train score = 0.87, Test score = 0.92



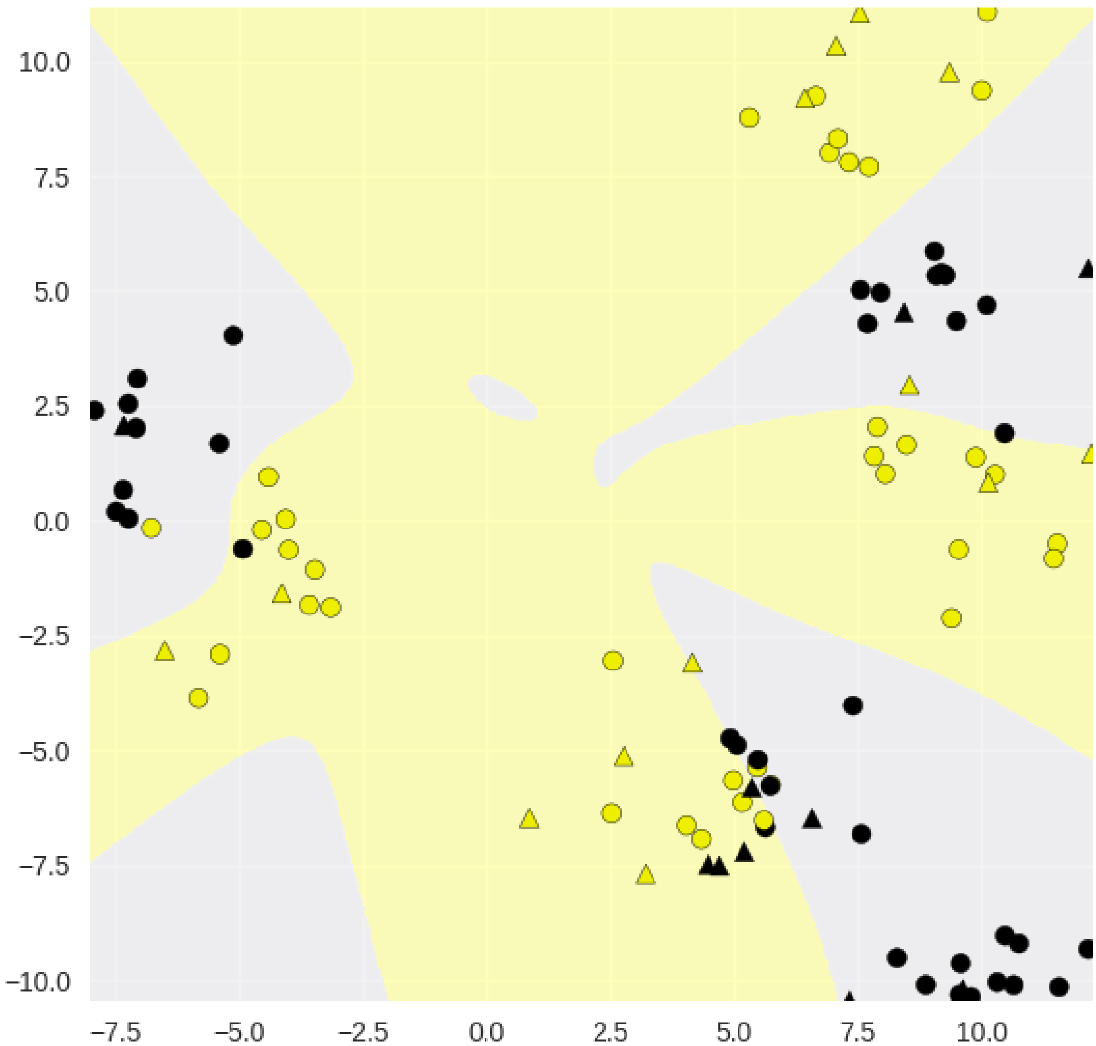


The effect of different choices of activation function

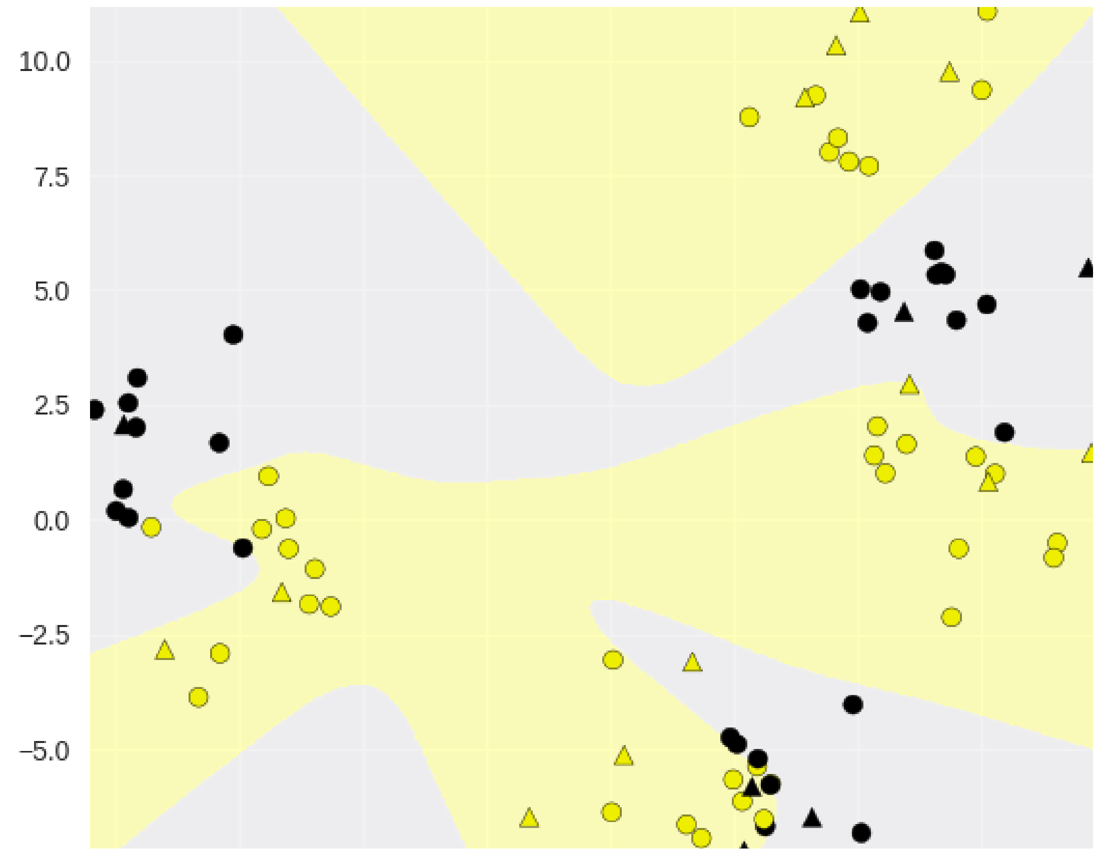
In [18]:

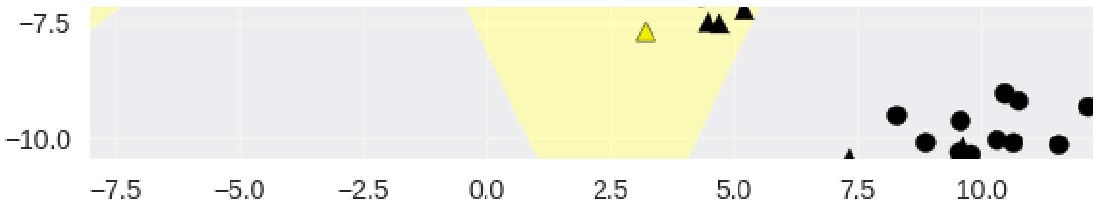
```
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_state=0)

fig, subaxes = plt.subplots(3, 1, figsize=(6,18))

for this_activation, axis in zip(['logistic', 'tanh', 'relu'], subaxes):
    nnclf = MLPClassifier(solver='lbfgs', activation = this_activation,
                         alpha = 0.1, hidden_layer_sizes = [10, 10],
                         random_state = 0).fit(X_train, y_train)

    title = 'Dataset 2: NN classifier, 2 layers 10/10, {} \
activation function'.format(this_activation)

    plot_class_regions_for_classifier_subplot(nnclf, X_train, y_train,
                                             X_test, y_test, title, axis)
    plt.tight_layout()
```

Dataset 2: NN classifier, 2 layers 10/10, logistic activation function
Train score = 0.92, Test score = 0.76

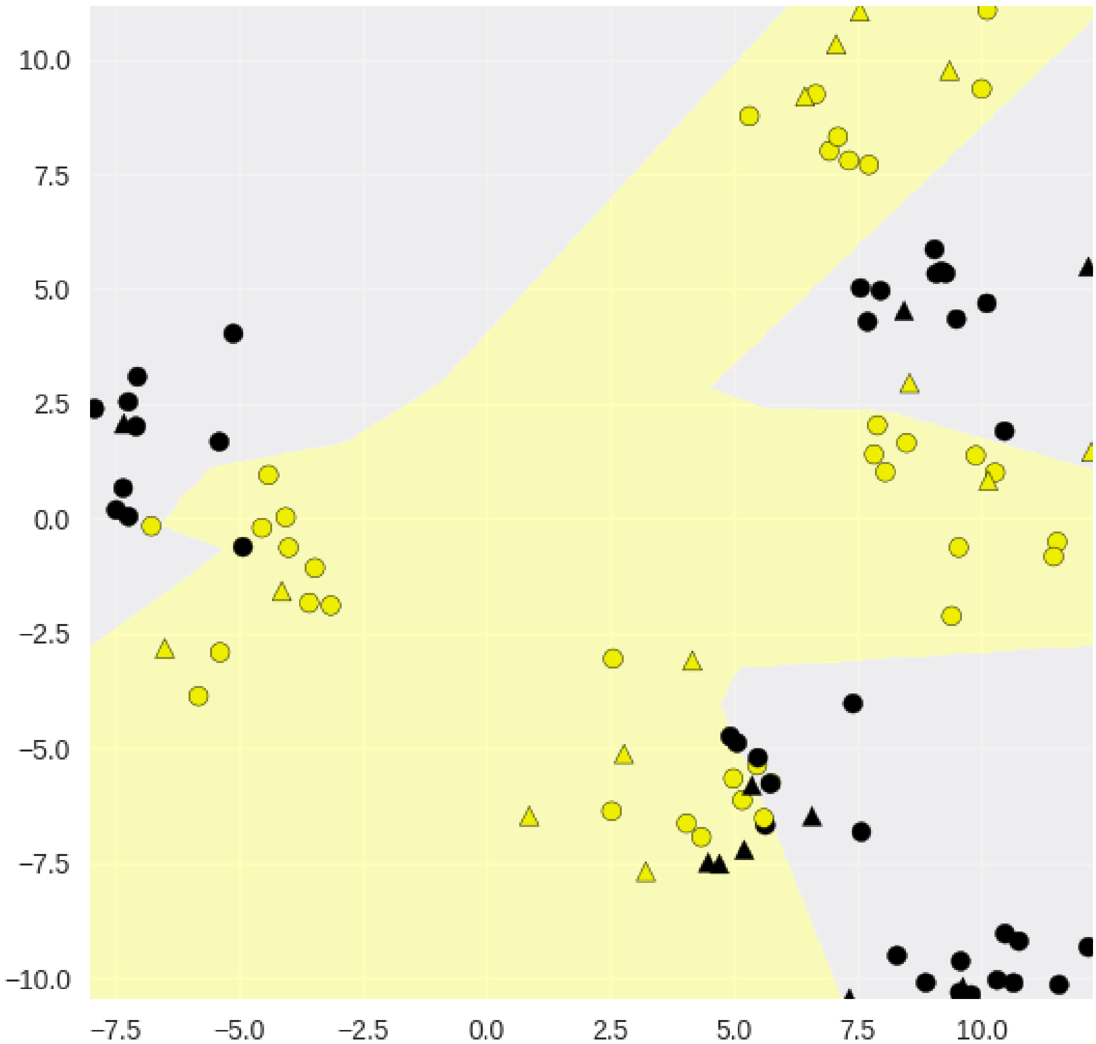


Dataset 2: NN classifier, 2 layers 10/10, tanh activation function
Train score = 0.93, Test score = 0.80





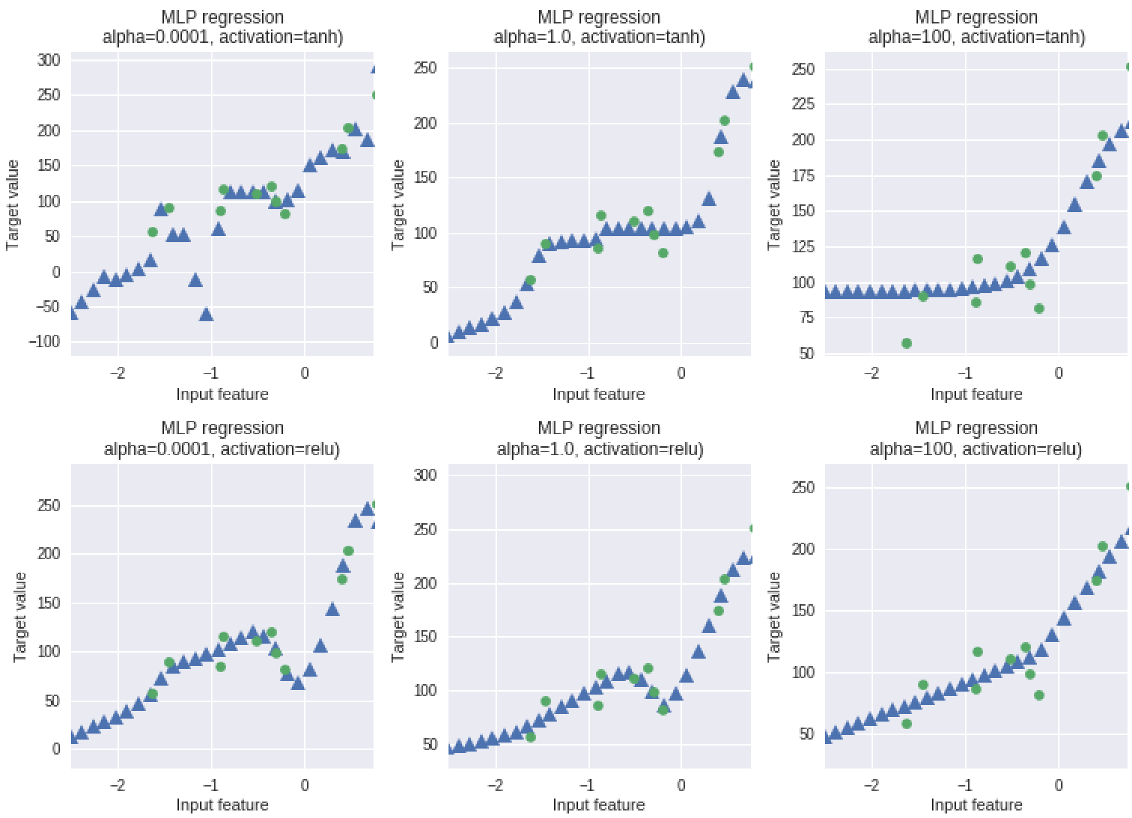
Dataset 2: NN classifier, 2 layers 10/10, relu activation function
Train score = 0.91, Test score = 0.76



Neural networks: Regression

In [17]:

```
from sklearn.neural_network import MLPRegressor # of course

fig, subaxes = plt.subplots(2, 3, figsize=(11,8), dpi=70)

X_predict_input = np.linspace(-3, 3, 50).reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X_R1[0::5], y_R1[0::5], rand
om_state = 0)

for thisaxisrow, thisactivation in zip(subaxes, ['tanh', 'relu']):
    for thisalpha, thisaxis in zip([0.0001, 1.0, 100], thisaxisrow):
        # Create the MLP Regression object and fit the training data
        # we create this MLP with 2 hidden layers of 100 nodes
        # a varied activation function
        # and also a varied alpha variable for L2 regularisation.
        mlpreg = MLPRegressor(hidden_layer_sizes = [100,100],
                              activation = thisactivation,
                              alpha = thisalpha,
                              solver = 'lbfgs').fit(X_train, y_train)
        y_predict_output = mlpreg.predict(X_predict_input)
        thisaxis.set_xlim([-2.5, 0.75])
        thisaxis.plot(X_predict_input, y_predict_output,
                     '^', markersize = 10)
        thisaxis.plot(X_train, y_train, 'o')
        thisaxis.set_xlabel('Input feature')
        thisaxis.set_ylabel('Target value')
        thisaxis.set_title('MLP regression\nalpha={}, activation={})'
                          .format(thisalpha, thisactivation))
        plt.tight_layout()
```



Application to real-world dataset for classification

In [16]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler # For preprocessing


scaler = MinMaxScaler() # To Normalise the Features

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_s
tate = 0)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = MLPClassifier(hidden_layer_sizes = [100, 100], alpha = 5.0,
                    random_state = 0, solver='lbfgs').fit(X_train_scaled, y_train
)

print('Breast cancer dataset')
print('Accuracy of NN classifier on training set: {:.2f}'
     .format(clf.score(X_train_scaled, y_train)))
print('Accuracy of NN classifier on test set: {:.2f}'
     .format(clf.score(X_test_scaled, y_test)))
```

```
Breast cancer dataset
Accuracy of NN classifier on training set: 0.98
Accuracy of NN classifier on test set: 0.97
```

04-05: Data Leakage

No sample code for this section.