Python ML - breast cancer diagnostic data set

Python notebook using data from Breast Cancer Wisconsin (Diagnostic) Data Set · 10,490 views · 2y ago

14 Fork 50

Version 5
5 commits

A brief tutorial on using Python to make predictions - Breast Cancer Wisconsin (Diagnostic) Data Set

de Freitas, R. C.

1 - Introduction

The aim of this notebook is to me (and others) to understand the process of organizing and preparing the data, selecting the features, choosing and applying the machine learning tools, comparing, selecting and improving the best models.

The features from the data set describe characteristics of the cell nuclei and are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. As described in UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29), the attribute informations are:

1. ID number
2. Diagnosis (M = malignant, B = benign)

3 - 32 Ten real-valued features are computed for each cell nucleus:

- a) radius (mean of distances from center to points on the perimeter)
- b) texture (standard deviation of gray-scale values)
- c) perimeter
- d) area
- e) smoothness (local variation in radius lengths)
- f) compactness (perimeter^2 / area - 1.0)
- g) concavity (severity of concave portions of the contour)
- h) concave points (number of concave portions of the contour)
- i) symmetry
- j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

2 - Preparing the data

We will start loading some of the packages that will help us organize and visualize the data. Other packages will be loaded as necessary.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

With help of Pandas (http://pandas.pydata.org/) we will load the data set and print some basic informations.

In [2]:

```
data = pd.read_csv('../input/data.csv');

print("\n \t The data frame has {0[0]} rows and {0[1]} column
s. \n".format(data.shape))
data.info()

data.head(3)
```

```
         The data frame has 569 rows and 33 columns.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
id                         569 non-null int64
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
Unnamed: 32                0 non-null float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smo
0	842302	M	17.99	10.38	122.8	1001.0	0.11
1	842517	M	20.57	17.77	132.9	1326.0	0.08
2	84300903	M	19.69	21.25	130.0	1203.0	0.10

3 rows × 33 columns

As can bee seen above, except for the diagnosis (that is M = malignant or B = benign) all other features are of type `float64` and have 0 non-null numbers.

During the data set loading a extra column was created. We will use the code below to delete this entire column.

In [3]:

```
data.drop(data.columns[[-1, 0]], axis=1, inplace=True)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

Now we can count how many diagnosis are malignant (M) and how many are benign (B). This is done below.

In [4]:

```
diagnosis_all = list(data.shape)[0]
diagnosis_categories = list(data['diagnosis'].value_counts())

print("\n \t The data has {} diagnosis, {} malignant and {} be
nign.".format(diagnosis_all,

diagnosis_categories[0],
diagnosis_categories[1]))
```

```
        The data has 569 diagnosis, 357 malignant and 212 ben
ign.
```

3 - Visualizing the data

In this section we will build visualizations of the data in order to decide how to proceed with the machine learning tools. To do that, we will need to use the Seaborn (https://seaborn.pydata.org/) and the Matplotlib (https://matplotlib.org/) packages.

We are interested mainly in the mean values of the features, so we will separate those features in the list below in order to make some work easier and the code more readably.

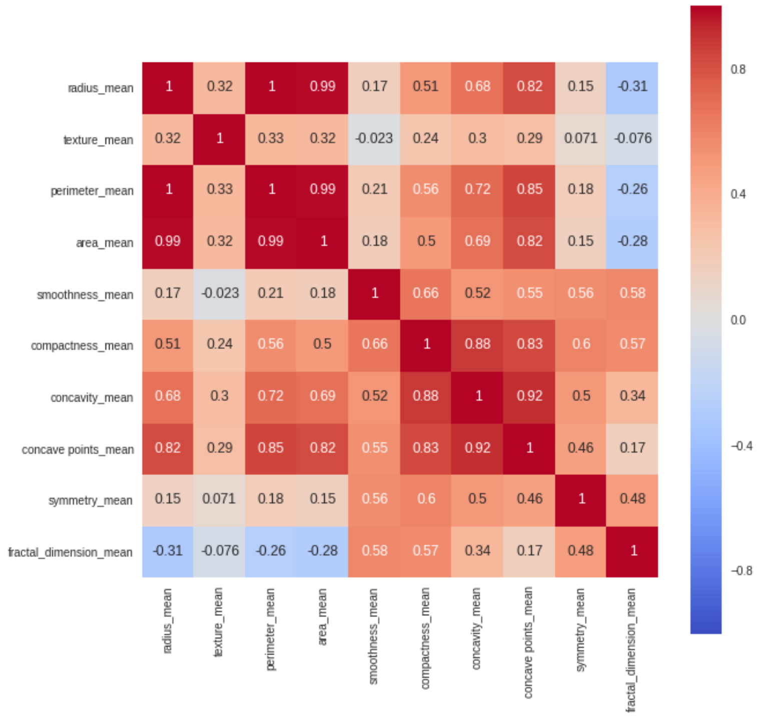
In [5]:

```
features_mean= list(data.columns[1:11])
```

Below we will use Seaborn to create a heat map of the correlations between the features.

In [6]:

```
plt.figure(figsize=(10,10))
sns.heatmap(data[features_mean].corr(), annot=True, square=Tru
e, cmap='coolwarm')
plt.show()
```



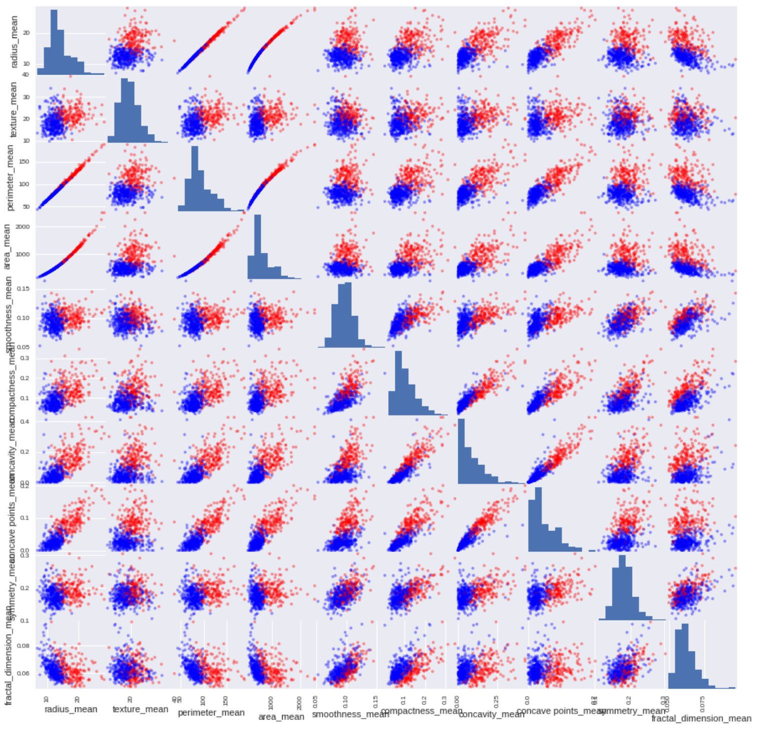
It is also possible to create a scatter matrix with the features. The red dots correspond to malignant diagnosis and blue to benign. Look how in some cases reds and blues dots occupies different regions of the plots.

In [7]:

```
color_dic = {'M':'red', 'B':'blue'}
```

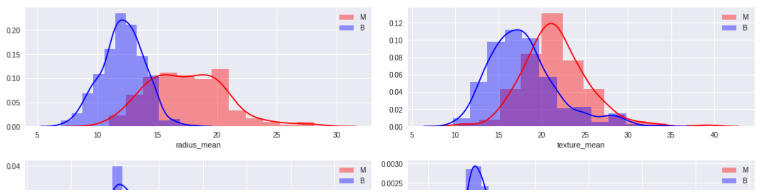
```
colors = data['diagnosis'].map(lambda x: color_dic.get(x))

sm = pd.scatter_matrix(data[features_mean], c=colors, alpha=0.
4, figsize=((15,15)));

plt.show()
```

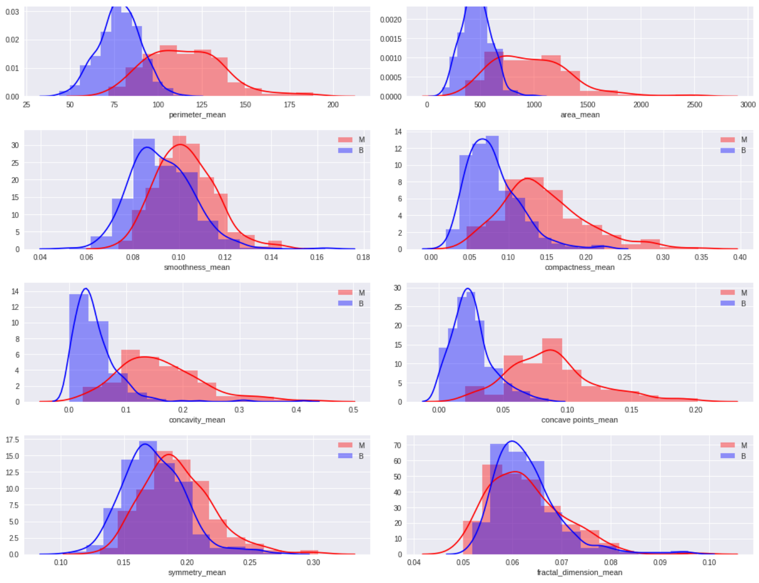
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:
4: FutureWarning: pandas.scatter_matrix is deprecated. Use pan
das.plotting.scatter_matrix instead
  after removing the cwd from sys.path.
```



We can also see how the malignant or benign tumors cells can have (or not) different values for the features plotting the distribution of each type of diagnosis for each of the mean features.

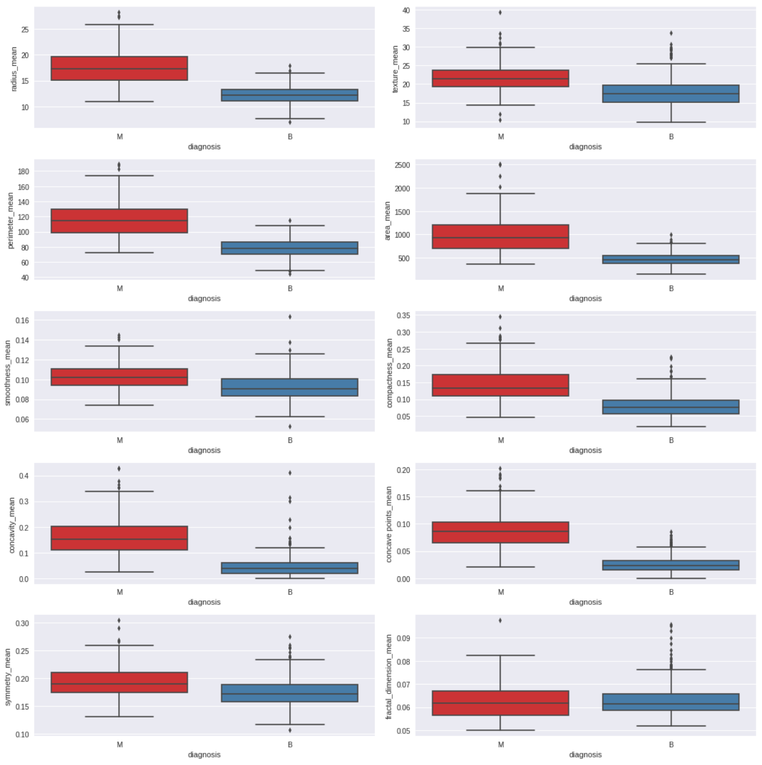
In [8]:
```
bins = 12
plt.figure(figsize=(15,15))
for i, feature in enumerate(features_mean):
    rows = int(len(features_mean)/2)

    plt.subplot(rows, 2, i+1)

    sns.distplot(data[data['diagnosis']=='M'][feature], bins=b
ins, color='red', label='M');
    sns.distplot(data[data['diagnosis']=='B'][feature], bins=b
ins, color='blue', label='B');

    plt.legend(loc='upper right')

plt.tight_layout()
plt.show()
```





Still another form of doing this could be using box plots, which is done below.

In [9]:
```
plt.figure(figsize=(15,15))
for i, feature in enumerate(features_mean):
    rows = int(len(features_mean)/2)

    plt.subplot(rows, 2, i+1)

    sns.boxplot(x='diagnosis', y=feature, data=data, palette=
"Set1")

plt.tight_layout()
plt.show()
```



As we saw above, some of the features can have, most of the times, values that will fall in some range depending on the diagnosis been malignant or benign. We will select those features to use in the next section the next section.

In [10]:
```
features_selection = ['radius_mean', 'perimeter_mean', 'area_m
ean', 'concavity_mean', 'concave points_mean']
```

4 - Machine learning

In this section we will test and analyze machine learning algorithms for classification in order to identify if the tumor is malignant or benign based on the cell features. For this we will use Scikit-learn (http://scikit-learn.org/stable/) package. The necessary tools will be loaded as needed.

The problem we are dealing with here is a classification problem. To choose the right estimator (algorithm) we used the flowchart (http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html) found in the Scikit-learn web page.

In [11]:
```
from sklearn.model_selection import train_test_split, cross_va
l_score
from sklearn.metrics import accuracy_score

import time
```

The algorithms will process only numerical values. For this reason, we will transform the categories M and B into values 1 and 0, respectively.

In [12]:
```
diag_map = {'M':1, 'B':0}
data['diagnosis'] = data['diagnosis'].map(diag_map)
```

4.1 - Using all mean values features

Our aim is to construct a "function" y = f(X) such that the value of y (1 or 0) will be determined once we input the values X into f. The "function" f will be construct by the machine learning algorithm based on the ys and Xs that are already known.

After training our machine learning algorithm we need to test its accuracy. In order to avoid Overfitting (https://en.wikipedia.org/wiki/Overfitting) we will use the function `train_test_split` to split the data randomly (`random_state = 42`) into a train and a test set. The test set will correspond to 20% of the total data (`test_size = 0.2`).

In [13]:
```
X = data.loc[:,features_mean]
y = data.loc[:, 'diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X, y, test
_size = 0.2, random_state = 42)

accuracy_all = []
cvs_all = []
```

Next we will use nine different classifiers, all with standard parameters. In all cases, the procedure will be the following:

1. the classifier `clf` is initialized;
2. the classifier `clf` is fitted with the train data set `X_train` and `y_train`;
3. the predictions are found using `X_test`;
4. the accuracy is estimated with help of cross-validation (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_
5. the accuracy (http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score) of the predictions is measured.

At the end the results are presents in %, along with the total time needed to run all the process.

4.1.1 - Stochastic Gradient Descent

The first classifier is the Stochastic Gradient Descent (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGI

In [14]:

```
from sklearn.linear_model import SGDClassifier

start = time.time()

clf = SGDClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("SGD Classifier Accuracy: {0:.2%}".format(accuracy_score
(prediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))
```

```
SGD Classifier Accuracy: 87.72%
Cross validation score: 76.59% (+/- 30.66%)
Execution time: 0.019594 seconds


/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/st
ochastic_gradient.py:84: FutureWarning: max_iter and tol param
eters have been added in <class 'sklearn.linear_model.stochast
ic_gradient.SGDClassifier'> in 0.19. If both are left unset, t
hey default to max_iter=5 and tol=None. If tol is not None, ma
x_iter defaults to max_iter=1000. From 0.21, default max_iter
will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

4.1.2 - **Support Vector Machines**

Now we will use three different Support Vector Machines (http://scikit-learn.org/stable/modules/svm.html) classifiers.

In [15]:

```
from sklearn.svm import SVC, NuSVC, LinearSVC

start = time.time()

clf = SVC()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("SVC Accuracy: {0:.2%}".format(accuracy_score(prediction
, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))

start = time.time()

clf = NuSVC()
clf.fit(X_train, y_train)
prediciton = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("NuSVC Accuracy: {0:.2%}".format(accuracy_score(predicti
on, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))

start = time.time()

clf = LinearSVC()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("LinearSVC Accuracy: {0:.2%}".format(accuracy_score(pred
iction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))
```

```
SVC Accuracy: 69.30%
Cross validation score: 71.70% (+/- 4.07%)
Execution time: 0.099833 seconds

NuSVC Accuracy: 69.30%
Cross validation score: 71.88% (+/- 3.97%)
Execution time: 0.10243 seconds

LinearSVC Accuracy: 77.19%
Cross validation score: 85.63% (+/- 12.86%)
```

```
Execution time: 0.17478 seconds
```

4.1.3 - Nearest Neighbors

The nearest neighbors classifier finds predefined number of training samples closest in distance to the new point, and predict the label from these.

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier

start = time.time()

clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_
test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))
```

```
Accuracy: 93.86%
Cross validation score: 88.60% (+/- 6.96%)
Execution time: 0.021457 seconds
```

4.1.3 - **Naive Bayes**

The Naive Bayes algorithm applies Bayes’ theorem with the assumption of independence between every pair of features.

In [17]:
```
from sklearn.naive_bayes import GaussianNB

start = time.time()

clf = GaussianNB()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_
test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
```

```
print("Execution time: {0:.5} seconds \n".format(end-start))
```

```
Accuracy: 94.74%
Cross validation score: 91.40% (+/- 5.03%)
Execution time: 0.015688 seconds
```

4.1.4 - Forest and tree methods

In [18]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier

start = time.time()

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("Random Forest Accuracy: {0:.2%}".format(accuracy_score(
prediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))

start = time.time()

clf = ExtraTreesClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("Extra Trees Accuracy: {0:.2%}".format(accuracy_score(pr
ediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))

start = time.time()

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))
```

```
print("Dedicion Tree Accuracy: {0:.2%}".format(accuracy_score(
prediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))
```

```
Random Forest Accuracy: 93.86%
Cross validation score: 93.69% (+/- 5.72%)
Execution time: 0.055068 seconds

Extra Trees Accuracy: 93.86%
Cross validation score: 93.16% (+/- 3.82%)
Execution time: 0.11165 seconds

Dedicion Tree Accuracy: 92.11%
Cross validation score: 91.58% (+/- 3.82%)
Execution time: 0.025084 seconds
```

4.2 - Using the selected features

In this section we will apply the same classifiers for the data with the features that were previously selected based on the analysis of section 3. To remember, those features are: radius_mean, perimeter_mean, area_mean, concavity_mean, concave points_mean.

In the end we will compare the accuracy the cross validation score for the selected set and the complete set of features.

In [19]:

```
X = data.loc[:,features_selection]
y = data.loc[:, 'diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test
_size = 0.2, random_state = 42)

accuracy_selection = []
cvs_selection = []
```

4.2.1 - Stochastic Gradient Descent

In [20]:

```
from sklearn.linear_model import SGDClassifier

start = time.time()

clf = SGDClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("SGD Classifier Accuracy: {0:.2%}".format(accuracy_score
(prediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
```

```
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))
```

```
SGD Classifier Accuracy: 81.58%
Cross validation score: 62.03% (+/- 17.10%)
Execution time: 0.01794 seconds
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/st
ochastic_gradient.py:84: FutureWarning: max_iter and tol param
eters have been added in <class 'sklearn.linear_model.stochast
ic_gradient.SGDClassifier'> in 0.19. If both are left unset, t
hey default to max_iter=5 and tol=None. If tol is not None, ma
x_iter defaults to max_iter=1000. From 0.21, default max_iter
will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

4.2.2 - Support Vector Machines

In [21]:

```
from sklearn.svm import SVC, NuSVC, LinearSVC

start = time.time()

clf = SVC()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("SVC Accuracy: {0:.2%}".format(accuracy_score(prediction
, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))

start = time.time()

clf = NuSVC()
clf.fit(X_train, y_train)
prediciton = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("NuSVC Accuracy: {0:.2%}".format(accuracy_score(predicti
on, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))

start = time.time()

clf = LinearSVC()
```

```
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("LinearSVC Accuracy: {0:.2%}".format(accuracy_score(pred
iction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))
```

```
SVC Accuracy: 74.56%
Cross validation score: 78.20% (+/- 8.67%)
Execution time: 0.09812 seconds

NuSVC Accuracy: 74.56%
Cross validation score: 80.49% (+/- 5.95%)
Execution time: 0.12121 seconds

LinearSVC Accuracy: 84.21%
Cross validation score: 69.91% (+/- 18.44%)
Execution time: 0.19853 seconds
```

4.2.3 - Nearest Neighbors

In [22]:

```
from sklearn.neighbors import KNeighborsClassifier

start = time.time()

clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_
test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))
```

```
Accuracy: 92.11%
Cross validation score: 88.25% (+/- 6.91%)
Execution time: 0.020711 seconds
```

4.2.4 - Naive Bayes

In [23]:

```
from sklearn.naive_bayes import GaussianNB

start = time.time()

clf = GaussianNB()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_
test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))
```

```
Accuracy: 94.74%
Cross validation score: 90.88% (+/- 5.83%)
Execution time: 0.019591 seconds
```

4.2.5 - Forest and tree methods

In [24]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier

start = time.time()

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("Random Forest Accuracy: {0:.2%}".format(accuracy_score(
prediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))

start = time.time()

clf = ExtraTreesClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))
```

```
print("Extra Trees Accuracy: {0:.2%}".format(accuracy_score(pr
ediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))

start = time.time()

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_selection.append(accuracy_score(prediction, y_test))
cvs_selection.append(np.mean(scores))

print("Dedicion Tree Accuracy: {0:.2%}".format(accuracy_score(
prediction, y_test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: %s seconds \n" % "{0:.5}".format(end-st
art))
```

```
Random Forest Accuracy: 92.11%
Cross validation score: 91.77% (+/- 7.26%)
Execution time: 0.15038 seconds

Extra Trees Accuracy: 92.11%
Cross validation score: 91.76% (+/- 4.91%)
Execution time: 0.11868 seconds

Dedicion Tree Accuracy: 88.60%
Cross validation score: 90.16% (+/- 4.08%)
Execution time: 0.024996 seconds
```

In [25]:

```
diff_accuracy = list(np.array(accuracy_selection) - np.array(a
ccuracy_all))
diff_cvs = list(np.array(cvs_selection) - np.array(cvs_all))

d = {'accuracy_all':accuracy_all, 'accuracy_selection':accurac
y_selection, 'diff_accuracy':diff_accuracy,
     'cvs_all':cvs_all, 'cvs_selection':cvs_selection, 'diff_c
vs':diff_cvs,}

index = ['SGD', 'SVC', 'NuSVC', 'LinearSVC', 'KNeighbors', 'Ga
ussianNB', 'RandomForest', 'ExtraTrees', 'DecisionTree']

df = pd.DataFrame(d, index=index)
```

In [26]:

```
df
```

Out[26]:

	accuracy_all	accuracy_selection	cvs_all	cvs_selection	diff_accuracy
SGD	0.877193	0.815789	0.765864	0.620285	-0.061404
SVC	0.692982	0.745614	0.717045	0.782008	0.052632
NuSVC	0.692982	0.745614	0.718815	0.804925	0.052632
LinearSVC	0.771930	0.842105	0.856252	0.699100	0.070175

KNeighbors	0.938596	0.921053	0.886002	0.882493	-0.017544
GaussianNB	0.947368	0.947368	0.914013	0.908765	0.000000
RandomForest	0.938596	0.921053	0.936930	0.917676	-0.017544
ExtraTrees	0.938596	0.921053	0.931558	0.917584	-0.017544
DecisionTree	0.921053	0.885965	0.915783	0.901593	-0.035088

As can be seen in the table above, using only some of the mean features reduced, in most of the cases, both accuracy and cross-validation scores.

5 - Improving the best model

Not all parameters of a classifier is learned from the estimators. Those parameters are called hyper-parameters and are passed as arguments to the constructor of the classifier. Each estimator has a different set of hyper-parameters, which can be found in the corresponding documentation.

We can search for the best performance of the classifier sampling different hyper-parameter combinations. This will be done with an exhaustive grid search (http://scikit-learn.org/stable/modules/grid_search.html#grid-search), provided by the GridSearchCV function.

The grid search will be done only on the best models, which are Naive Bayes, Random Forest, Extra Trees and Decision Trees.

After running the piece of codes below, it will be presented the accuracy, the cross-validation score and the best set of parameters.

In [27]:

```
from sklearn.model_selection import GridSearchCV

X = data.loc[:,features_mean]
y = data.loc[:, 'diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test
_size = 0.2, random_state = 42)

accuracy_all = []
csv_all = []
```

5.1 - Naive Bayes

In [28]:

```
start = time.time()

parameters = {'priors':[[0.01, 0.99],[0.1, 0.9], [0.2, 0.8], [
0.25, 0.75], [0.3, 0.7],[0.35, 0.65], [0.4, 0.6]]}

clf = GridSearchCV(GaussianNB(), parameters, scoring = 'averag
e_precision', n_jobs=-1)
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))

print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_
test)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(n
p.mean(scores), np.std(scores)*2))
print("Execution time: {0:.5} seconds \n".format(end-start))

print("Best parameters: {0}".format(clf.best_params_))
```

```
Accuracy: 94.74%
Cross validation score: 95.45% (+/- 9.20%)
Execution time: 4.2847 seconds

Best parameters: {'priors': [0.1, 0.9]}
```

5.2 - Forest and tree methods

In [29]:

```
start = time.time()

parameters = {'n_estimators':list(range(1,101)), 'criterion':[
'gini', 'entropy']}

clf = GridSearchCV(RandomForestClassifier(), parameters, scori
ng = 'average_precision', n_jobs=-1)
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=5)

end = time.time()

accuracy_all.append(accuracy_score(prediction, y_test))
cvs_all.append(np.mean(scores))
```

Did you find this Kernel useful?
Show your appreciation with an upvote

14

Data

Data Sources

Breast Cancer Wisc...

d... 32 columns

Breast Cancer Wisconsin (Diagnostic) Data Set

Predict whether the cancer is benign or malignant

Last Updated: 3 years ago (Version 2)

About this Dataset

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of

Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server: ftp ftp.cs.wisc.edu cd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

Attribute Information:

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-scale values) c) perimeter d) area e) smoothness (local variation in radius lengths) f) compactness (perimeter^2 / area - 1.0) g) concavity (severity of concave portions of the contour) h) concave points (number of concave portions of the contour) i) symmetry j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the

Run Info

Succeeded	True	Run Time	1256.3 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python (Dockerfile)	Output Size	0
		Used All Space	False
Timeout Exceeded	False		
Failure Message			

Log

Download Log

```
Time  Line #  Log Message
           1  [{
           2    "data": "[NbConvertApp] Converting notebook
              __notebook_source__.ipynb to html\n",
           3    "stream_name": "stderr",
           4    "time": 2.4531012009829283
           5  },{
           6    "data": "[NbConvertApp] ERROR | Notebook JSON is invalid:
              Additional properties are not allowed ('execution_count',
              'outputs' were unexpected)\n\nFailed validating
              'additionalProperties' in markdown_cell:\n\nOn
              instance['cells'][0]:\n{'cell_type': 'markdown',\n
              'execution_count': None,\n 'metadata': {'_cell_guid':
              '9238e5cc-2caf-4093-ad4c-b1dea9dc9323',\n
              '_execution_state': 'idle',\n              '_uuid':
              '9f3982b97d5b5a01181484a6026cb4cd5382c73b',\n
              'collapsed': False},\n 'outputs': ['...0 outputs...'],\n
              'source': '# **A brief tutorial on using Python to make
              predictions - '\n            'Breas...'}\n",
           7    "stream_name": "stderr",
           8    "time": 2.4588050998281687
           9  },{
          10    "data": "[NbConvertApp] Writing 377965 bytes to
              __results__.html\n",
          11    "stream_name": "stderr",
          12    "time": 2.748434484936297
          13  }{
          14    "data": "[NbConvertApp] Converting notebook
              __notebook_source__.ipynb to notebook\n",
          15    "stream_name": "stderr",
          16    "time": 2.5452729507815093
          17  },{
          18    "data": "[NbConvertApp] ERROR | Notebook JSON is invalid:
              Additional properties are not allowed ('outputs',
              'execution_count' were unexpected)\n\nFailed validating
              'additionalProperties' in markdown_cell:\n\nOn
              instance['cells'][0]:\n{'cell_type': 'markdown',\n
              'execution_count': None,\n 'metadata': {'_cell_guid':
              '9238e5cc-2caf-4093-ad4c-b1dea9dc9323',\n
'_execution_state': 'idle',\n              '_uuid':
'9f3982b97d5b5a01181484a6026cb4cd5382c73b',\n
'collapsed': False},\n 'outputs': ['...0 outputs...'],\n
'source': '# **A brief tutorial on using Python to make
predictions - '\n           'Breas...'}\n[NbConvertApp]
Executing notebook with kernel: python3\n",
  "stream_name": "stderr",
  "time": 2.592556295916438
},{
  "data": "Fontconfig warning: ignoring C.UTF-8: not a valid
language tag\n",
  "stream_name": "stderr",
  "time": 4.821897581918165
},{
  "data": "[NbConvertApp] ERROR | Notebook JSON is invalid:
Additional properties are not allowed ('outputs',
'execution_count' were unexpected)\n\nFailed validating
'additionalProperties' in markdown_cell:\n\nOn
instance['cells'][0]:\n{'cell_type': 'markdown',\n
'execution_count': None,\n 'metadata': {'_cell_guid':
'9238e5cc-2caf-4093-ad4c-b1dea9dc9323',\n
'_execution_state': 'idle',\n              '_uuid':
'9f3982b97d5b5a01181484a6026cb4cd5382c73b',\n
'collapsed': False},\n 'outputs': ['...0 outputs...'],\n
'source': '# **A brief tutorial on using Python to make
predictions - '\n           'Breas...'}\n",
  "stream_name": "stderr",
  "time": 451.81696302397177
},{
  "data": "[NbConvertApp] Writing 1332813 bytes to
__notebook__.ipynb\n",
  "stream_name": "stderr",
  "time": 451.82994684786536
}{
  "data": "[NbConvertApp] Converting notebook
__notebook__.ipynb to html\n",
  "stream_name": "stderr",
  "time": 2.2361925679724663
},{
  "data": "[NbConvertApp] ERROR | Notebook JSON is invalid:
Additional properties are not allowed ('execution_count',
'outputs' were unexpected)\n\nFailed validating
'additionalProperties' in markdown_cell:\n\nOn
instance['cells'][0]:\n{'cell_type': 'markdown',\n
'execution_count': None,\n 'metadata': {'_cell_guid':
'9238e5cc-2caf-4093-ad4c-b1dea9dc9323',\n
'_execution_state': 'idle',\n              '_uuid':
'9f3982b97d5b5a01181484a6026cb4cd5382c73b',\n
'collapsed': False},\n 'outputs': ['...0 outputs...'],\n
'source': '# **A brief tutorial on using Python to make
predictions - '\n           'Breas...'}\n",
  "stream_name": "stderr",
  "time": 2.251680816989392
},{
  "data": "[NbConvertApp] Support files will be in
__results___files/\n[NbConvertApp] Making directory
__results___files\n",
  "stream_name": "stderr",
  "time": 2.542066178051755
},{
  "data": "[NbConvertApp] Making directory
__results___files\n[NbConvertApp] Making directory
__results___files\n[NbConvertApp] Making directory
__results___files\n[NbConvertApp] Writing 395040 bytes to
__results__.html\n",
  "stream_name": "stderr",
  "time": 2.5463952941354364
}

Complete. Exited with code 0.
```

Comments (0)

Click here to comment...