# PatientClassifierv1

March 4, 2019

In [364]:
```
##########################################################################################
### A 10,000-patient database that contains in total 10,000 patients, 36,143 admissi
##########################################################################################



###################################
### PatientCorePopulatedTable ###
###################################
#
#[PatientID] - a unique ID representing a patient.
#[PatientGender] - Male/Female.
#[PatientDateOfBirth] - Date Of Birth.
#[PatientRace] - African American, Asian, White.
#[PatientMaritalStatus] - Single, Married, Divorced, Separated, Widowed.
#[PatientLanguage] - English, Icelandic, Spanish.
#[PatientPopulationPercentageBelowPoverty] - given in %.
#
#####################################
### AdmissionsCorePopulatedTable ###
#####################################
#
#[PatientID] - a unique ID representing a patient.
#[AdmissionID] - an admission ID for the patient.
#[AdmissionStartDate] - start date.
#[AdmissionEndDate] - end date.
#
#############################################
### AdmissionsDiagnosesCorePopulatedTable ###
#############################################
#
#[PatientID] - a unique ID representing a patient.
#[AdmissionID] - an admission ID for the patient.
#[PrimaryDiagnosisCode] - ICD10 code for admission's primary diagnosis.
#[PrimaryDiagnosisDescription] - admission's primary diagnosis description.
#
##############################
```

```
### LabsCorePopulatedTable ###
##############################
#
#[PatientID] - a unique ID representing a patient.
#[AdmissionID] - an admission ID for the patient.
#[LabName] - lab's name, including:
#[LabValue] - lab's value
#[LabUnits] - lab's units.
#[LabDateTime] - date.
```

In [47]:
```
from __future__ import print_function

print(__doc__)

import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import decomposition
from sklearn import datasets
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
from scipy.ndimage import convolve
from sklearn import linear_model, datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.neural_network import BernoulliRBM
from sklearn.pipeline import Pipeline
from sklearn.base import clone
from sklearn.datasets import make_multilabel_classification
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import CCA
from matplotlib.mlab import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from scipy.stats import ttest_ind
from sklearn.gaussian_process import GaussianProcessClassifier
```

```
from sklearn.gaussian_process.kernels import RBF
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
from sklearn.utils import shuffle
from sklearn import tree
#from sklearn.cross_validation import cross_val_score
from pydotplus import graph_from_dot_data
from sklearn.linear_model import LogisticRegression
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
from sklearn import  linear_model
from sklearn.model_selection import cross_val_score
from IPython.display import display
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.calibration import calibration_curve
from sklearn import neighbors
from sklearn.calibration import calibration_curve
from sklearn.calibration import CalibratedClassifierCV
import seaborn as sns
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.metrics import classification_report,confusion_matrix
#from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
import scipy.stats as stats
#import researchpy as rp
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.pyplot as plt
from IPython.display import Image
```

```
Automatically created module for IPython interactive environment
```

In [2]:
```
##################################
### PatientCorePopulatedTable ###
##################################


file_to_open = "AdmissionsCorePopulatedTable.csv"
columnsadmit = ['PatientID', 'AdmissionID', 'AdmissionStartDate', 'AdmissionEndDate']
f = open(file_to_open)
admitdf = pd.read_csv(f, index_col=False, names=columnsadmit)
admitdf.head()
```

In [3]:
```
#####################################
### AdmissionsCorePopulatedTable ###
#####################################


file_to_open1 =  "PatientCorePopulatedTable.csv"
columnspatient = ['PatientID', 'PatientGender', 'PatientDateOfBirth', 'PatientRace', '
f1 = open(file_to_open1)
patientdf = pd.read_csv(f1, index_col=False, names=columnspatient)
patientdf.head()
```

In [4]:
```
###############################################
### AdmissionsDiagnosesCorePopulatedTable ###
###############################################

file_to_open2 =  "AdmissionsDiagnosesCorePopulatedTable.csv"
columnsdiagnoses = ['PatientID', 'PatientGender', 'PrimaryDiagnosisCode', 'PrimaryDiagn
f2 = open(file_to_open2)
```

```
        diagnosesdf = pd.read_csv(f2, index_col=False, names=columnsdiagnoses)
        del diagnosesdf['PatientMaritalStatus']
        diagnosesdf.head()

In [5]: ##############################
        ### LabsCorePopulatedTable ###
        ##############################
        file_to_open3 = "1.txt"
        columnslabs = ['PatientID', 'AdmissionID', 'LabName', '[LabValue]','[LabUnits]','[LabDa
        df3 = open(file_to_open3)
        labsdf1 = pd.read_csv(df3, index_col=False , names=columnslabs,sep="\t", engine='pythor




        labsdf1.head()

In [6]: pulmonarydisdf=diagnosesdf[diagnosesdf.PrimaryDiagnosisCode.str.startswith('I2')]
        pulmonarydisdf

In [0]:

In [7]: frames = labsdf1
        result = frames
        result=result.drop([0])

In [8]: result.head()

In [0]:

In [9]: #check
        presults = pd.merge(result, pulmonarydisdf[['PatientID','PatientGender','PrimaryDiagnos
        presults

In [10]: results = pd.merge(result, diagnosesdf[['PatientID', 'PatientGender', 'PrimaryDiagnosi
         results.head()

In [0]:

In [0]:

In [11]: # Select samples at random from
         results = results.sample(n=6000)
         results

In [12]: frames = [results, presults]
         results = pd.concat(frames)
         results.tail()

In [13]: results.loc[results['PrimaryDiagnosisCode'].str.startswith('I2'), 'PrimaryDiagnosisCo
         results.loc[results['PrimaryDiagnosisCode'].str.startswith('I2')==False, 'PrimaryDiagn
```

```
In [14]: y=results['PrimaryDiagnosisCode']
         y=y.astype('int')

In [15]: list(results)

Out[15]: ['PatientID',
          'AdmissionID',
          'LabName',
          '[LabValue]',
          '[LabUnits]',
          '[LabDateTime]',
          'PatientGender',
          'PrimaryDiagnosisCode',
          'PrimaryDiagnosisDescription']

In [16]: results.head()

In [17]: classifiers = [
             KNeighborsClassifier(3),
             SVC(kernel="linear", C=0.025),
             SVC(gamma='scale', 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(),
             QuadraticDiscriminantAnalysis()]

In [18]: results = pd.merge(results, patientdf[['PatientID',        'PatientGender',        'Pa

In [19]: results.head()

In [20]: one_hot_PatientRace = pd.get_dummies(results['PatientRace'])
         results=results.join(one_hot_PatientRace)

In [21]: results = results.drop(['PatientID','AdmissionID','LabName','[LabUnits]','PatientGende

In [22]: results.head()

In [23]: results['[LabDateTime]'] = pd.to_datetime(results['[LabDateTime]'])
         results['year'], results['month'] = results['[LabDateTime]'].dt.year, results['[LabDat

In [24]: results = results.drop(['PatientDateOfBirth','month','[LabDateTime]'], axis=1)

In [0]:

In [25]: list(results)
```

```
Out[25]: ['[LabValue]',
          'PatientGender_x',
          'PrimaryDiagnosisCode',
          'PatientPopulationPercentageBelowPoverty',
          'African American',
          'Asian',
          'Unknown',
          'White',
          'year']

In [0]:

In [26]: x=results.drop('PrimaryDiagnosisCode',axis=1)


In [27]: # Randomly, split the data into test/training/validation sets
         x_train, x_test, x_validate = np.split(results.sample(frac=1), [int(.6*len(results)),
         print(x_train.shape,x_test.shape,x_validate.shape)

(56096, 9) (18699, 9) (18699, 9)


In [28]: #Randomly, split the data into test/training/validation sets
         y_train, y_test, y_validate = np.split(y.sample(frac=1), [int(.6*len(results)), int(.8
         print(y_train.shape, y_test.shape, y_validate.shape)

(56096,) (18699,) (18699,)


In [29]: y_train

Out[29]: 64930      1
         76928      1
         73518      1
         29470      1
         2209       1
         7430       1
         55693      1
         5172       1
         19958      1
         41917      1
         9786       1
         2617794    0
         86044      1
         16341      1
         51311      1
         4321       1
         22593      1
         73154      1
```

```
29388      1
72761      1
53630      1
7988       1
73265      1
71130      1
36340      1
1231       1
56691      1
74791      1
78443      1
86087      1
          ..
26791      1
50261      1
40999      1
55140      1
39826      1
34582      1
77297      1
570013     0
9115       1
57466      1
49446      1
74011      1
17202      1
30623      1
47504      1
39880      1
73800      1
14697      1
2413186    0
47264      1
7590       1
33884      1
84506      1
6307       1
30821      1
82461      1
52968      1
3587639    0
73498      1
39365      1
Name: PrimaryDiagnosisCode, Length: 56096, dtype: int64
```

In [30]:
```
# Check the balance of the splits on y_
y_test.mean()
```

Out[30]:
```
0.9366276271458367
```

```
In [31]: y_train.mean()
```

```
Out[31]: 0.937375213918996
```

```
In [32]: ols = linear_model.LinearRegression()
         model = ols.fit(x_train, y_train)

         print(model.predict(x_test))
```

```
[0.9383396  0.93999641 0.93829231 ... 0.93336125 0.93510995 0.93072031]
```

```
In [33]: model.score(x_test, y_test)
```

```
Out[33]: -0.0005430119568730074
```

```
In [34]: # Variable importance
         rf = RandomForestClassifier()
         rf.fit(x_train, y_train)
         print ("Features sorted by their score:" )

         print(sorted(zip(map(lambda x: round(x, 4), rf.feature_importances_), x_train), revers
```

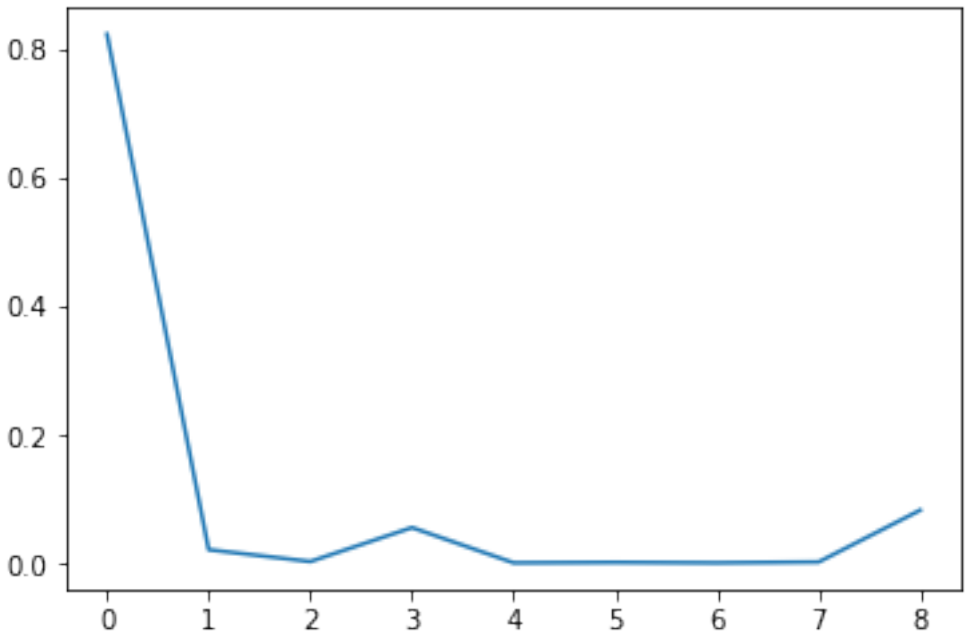
```
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/ensemble/forest.py:246: Futu
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
Features sorted by their score:
[(0.8229, '[LabValue]'), (0.0838, 'year'), (0.0568, 'PatientPopulationPercentageBelowPoverty')
```

```
In [35]: plt.plot( rf.feature_importances_)
```

```
Out[35]: [<matplotlib.lines.Line2D object at 0x7f4ebe180850>]
```

```
Out[35]:
```




In [36]:
```
# Instantiate
logit_model = LogisticRegression()
# Fit
logit_model = logit_model.fit(x_train, y_train)
# How accurate?
logit_model.score(x_train, y_train)
#0.7874
```

```
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/linear_model/logistic.py:433
  FutureWarning)
```

Out[36]: 0.937375213918996

In [37]:
```
#PrimaryDiagnosisCode.str.startswith('I2')
#results.loc[[results.PrimaryDiagnosisCode.str.startswith('I2')],results['PrimaryDiag
# How does it perform on the test dataset?

# Predictions on the test dataset
predicted = pd.DataFrame(logit_model.predict(x_test))
# Probabilities on the test dataset
probs = pd.DataFrame(logit_model.predict_proba(x_test))
print (metrics.accuracy_score(y_test, predicted))
```

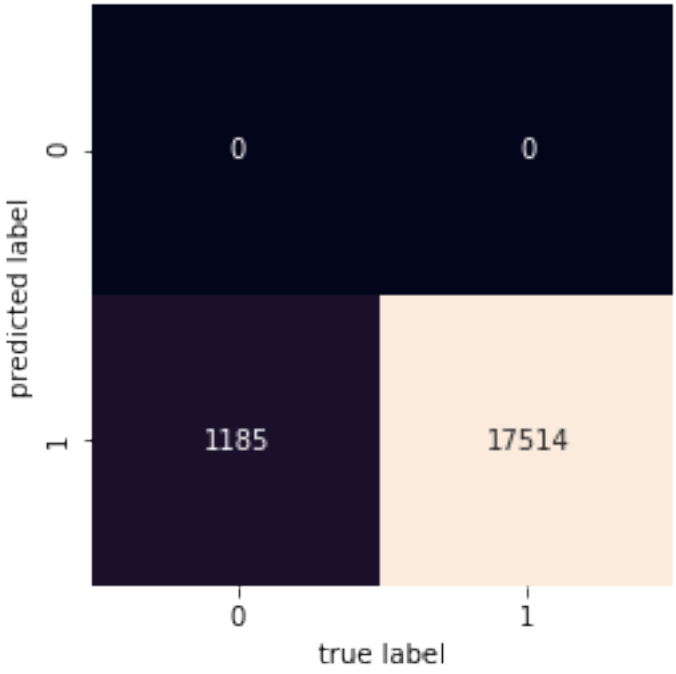
```
0.9366276271458367
```

```
In [38]: print(metrics.confusion_matrix(y_test, predicted))
```

```
[[    0  1185]
 [    0 17514]]
```

```
In [39]: from sklearn.metrics import confusion_matrix
         mat = confusion_matrix(y_test, predicted)
         sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
         plt.xlabel('true label')
         plt.ylabel('predicted label');
```

Out[39]:




```
In [40]: print( metrics.classification_report(y_test, predicted))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1185
           1       0.94      1.00      0.97     17514

   micro avg       0.94      0.94      0.94     18699
   macro avg       0.47      0.50      0.48     18699
weighted avg       0.88      0.94      0.91     18699
```

```
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
```

In [41]:
```
# Instantiate with a max depth of 3
tree_model = tree.DecisionTreeClassifier(max_depth=3)
# Fit a decision tree
tree_model = tree_model.fit(x_train, y_train)
# Training accuracy
tree_model.score(x_train, y_train)
```

Out[41]:
```
0.9374108670849971
```

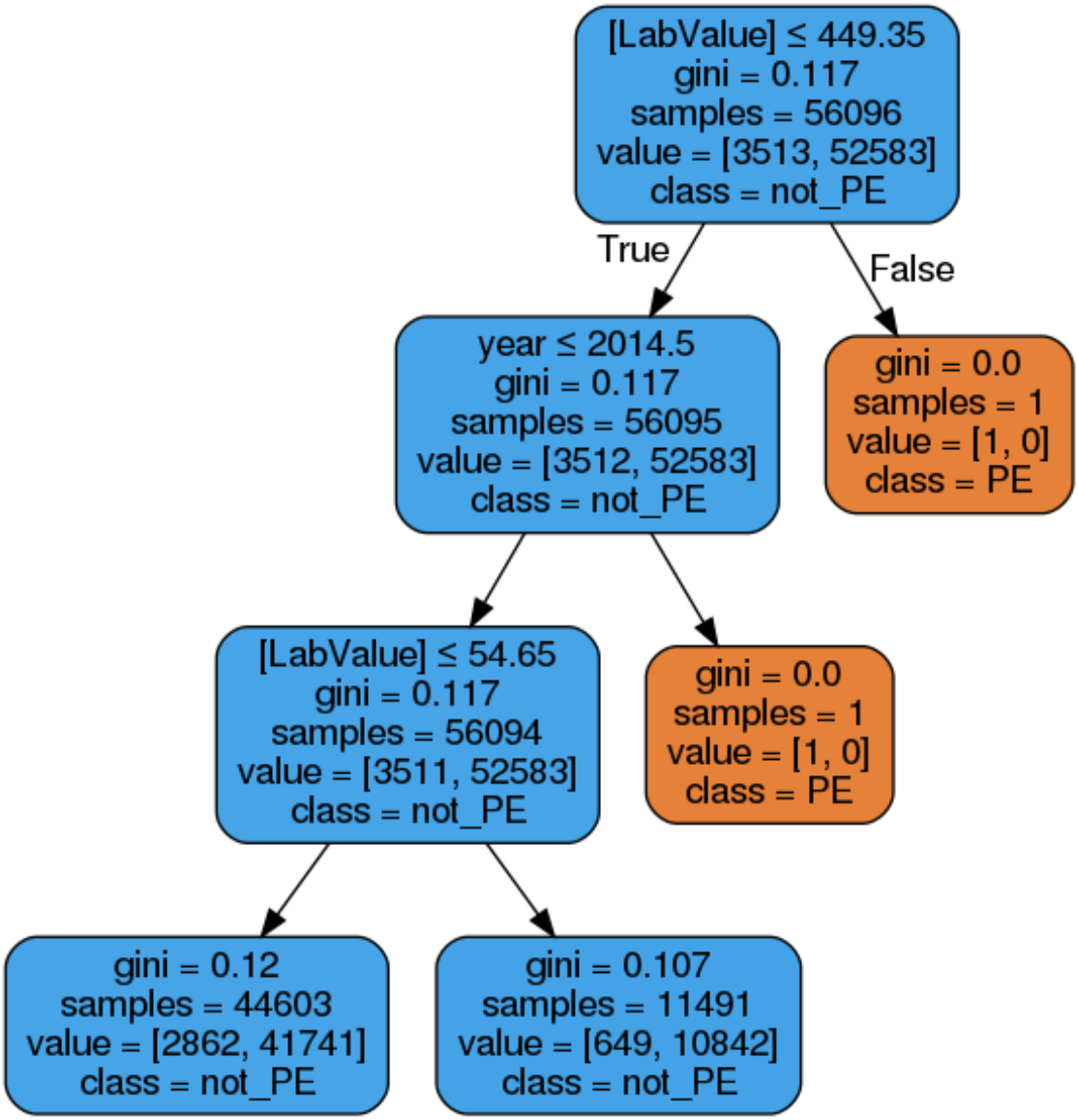
In [42]:
```
# Predictions/probs on the test dataset
predicted = pd.DataFrame(tree_model.predict(x_test))
probs = pd.DataFrame(tree_model.predict_proba(x_test))
```

In [43]:
```
# Store metrics
tree_accuracy = metrics.accuracy_score(y_test, predicted)
tree_roc_auc = metrics.roc_auc_score(y_test, probs[1])
tree_confus_matrix = metrics.confusion_matrix(y_test, predicted)
tree_classification_report = metrics.classification_report(y_test, predicted)
tree_precision = metrics.precision_score(y_test, predicted, pos_label=1)
tree_recall = metrics.recall_score(y_test, predicted, pos_label=1)
tree_f1 = metrics.f1_score(y_test, predicted, pos_label=1)
```

In [0]:

In [49]:
```
# evaluate the model using 10-fold cross-validation
tree_cv_scores = cross_val_score(tree.DecisionTreeClassifier(max_depth=3), x_test, y_t

# output decision plot
dot_data = tree.export_graphviz(tree_model, out_file=None,
                        feature_names=x_test.columns.tolist(),
                        class_names=['PE', 'not_PE'],
                        filled=True, rounded=True,
                        special_characters=True)
graph = graph_from_dot_data(dot_data)
graph.write_png("decision_treerun5.png")
Image("decision_treerun5.png")
```

Out[49]:




In [50]:
```
# Instantiate
rf = RandomForestClassifier()
# Fit
rf_model = rf.fit(x_train, y_train)
# training accuracy 99.74%
rf_model.score(x_train, y_train)
print(rf_model.score(x_train, y_train))
```

```
0.970907016543069
```

In [51]:
```
# Predictions/probs on the test dataset
```

```
predicted = pd.DataFrame(rf_model.predict(x_test))
probs = pd.DataFrame(rf_model.predict_proba(x_test))
```

In [52]:
```
# Store metrics
rf_accuracy = metrics.accuracy_score(y_test, predicted)
rf_roc_auc = metrics.roc_auc_score(y_test, probs[1])
rf_confus_matrix = metrics.confusion_matrix(y_test, predicted)
rf_classification_report = metrics.classification_report(y_test, predicted)
rf_precision = metrics.precision_score(y_test, predicted, pos_label=1)
rf_recall = metrics.recall_score(y_test, predicted, pos_label=1)
rf_f1 = metrics.f1_score(y_test, predicted, pos_label=1)
```

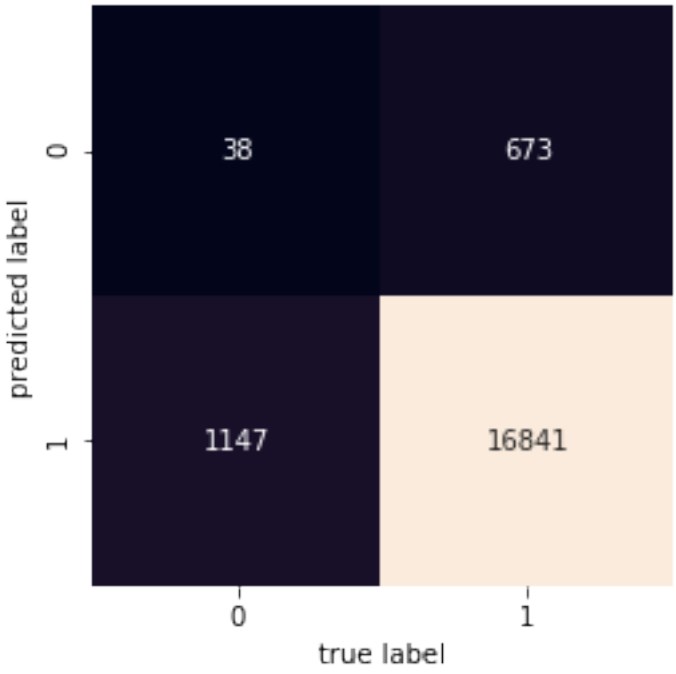
In [53]:
```
print(rf_confus_matrix)
```

```
[[   38  1147]
 [  673 16841]]
```

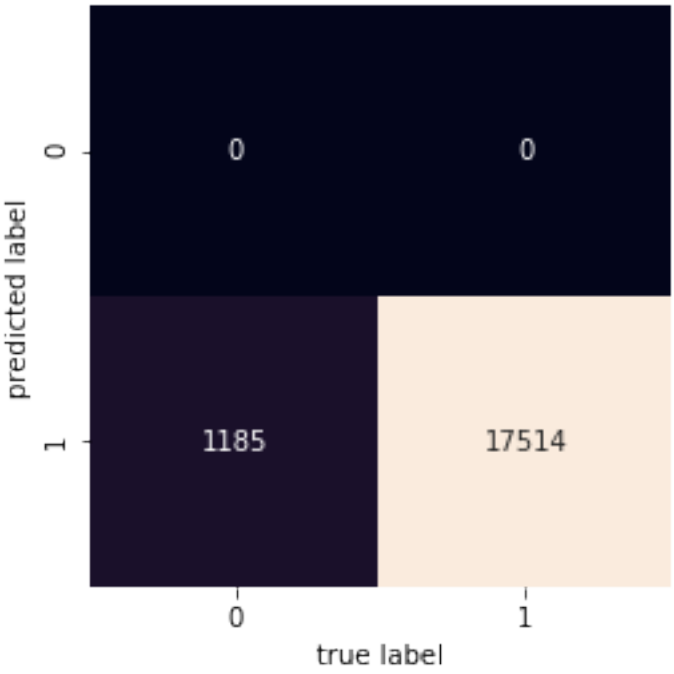
In [54]:
```
mat = confusion_matrix(y_test, predicted)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[54]:
```
Text(91.68,0.5,'predicted label')
```

Out[54]:



```
In [55]: # Instantiate
         svm_model = SVC(probability=True, gamma='scale')
         # Fit
         svm_model = svm_model.fit(x_train, y_train)
         # Accuracy
         svm_model.score(x_train, y_train)

         # Predictions/probs on the test dataset
         predicted = pd.DataFrame(svm_model.predict(x_test))
         probs = pd.DataFrame(svm_model.predict_proba(x_test))

         # Store metrics
         svm_accuracy = metrics.accuracy_score(y_test, predicted)
         svm_roc_auc = metrics.roc_auc_score(y_test, probs[1])
         svm_confus_matrix = metrics.confusion_matrix(y_test, predicted)
         svm_classification_report = metrics.classification_report(y_test, predicted)
         svm_precision = metrics.precision_score(y_test, predicted, pos_label=1)
         svm_recall = metrics.recall_score(y_test, predicted, pos_label=1)
         svm_f1 = metrics.f1_score(y_test, predicted, pos_label=1)

         #Evaluate the model using 10-fold cross-validation
         #svm_cv_scores = cross_val_score(SVC(probability=True), x_test, y_test, scoring='prec
         #svm_cv_mean = np.mean(svm_cv_scores)

/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)


In [56]: print(svm_confus_matrix)

[[    0  1185]
 [    0 17514]]


In [57]: mat = confusion_matrix(y_test, predicted)
         sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
         plt.xlabel('true label')
         plt.ylabel('predicted label')

Out[57]: Text(91.68,0.5,'predicted label')

Out[57]:
```




In [58]:
```
# instantiate learning model (k = 3)
knn_model = KNeighborsClassifier(n_neighbors=int(3))
# fit the model
knn_model.fit(x_train, y_train)
# Accuracy
knn_model.score(x_train, y_train)
```

Out[58]: 0.9405661722760981

In [59]:
```
# Predictions/probs on the test dataset
predicted = pd.DataFrame(knn_model.predict(x_test))
probs = pd.DataFrame(knn_model.predict_proba(x_test))
```

In [60]:
```
# Store metrics
knn_accuracy = metrics.accuracy_score(y_test, predicted)
knn_roc_auc = metrics.roc_auc_score(y_test, probs[1])
knn_confus_matrix = metrics.confusion_matrix(y_test, predicted)
knn_classification_report = metrics.classification_report(y_test, predicted)
knn_precision = metrics.precision_score(y_test, predicted, pos_label=1)
knn_recall = metrics.recall_score(y_test, predicted, pos_label=1)
knn_f1 = metrics.f1_score(y_test, predicted, pos_label=1)
```

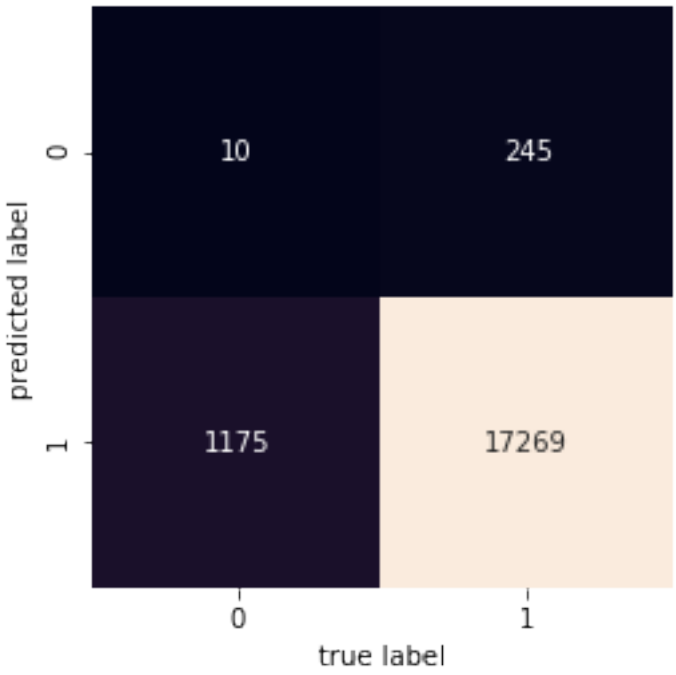
In [61]:
```
print(knn_confus_matrix)
```

```
[[   10  1175]
 [  245 17269]]
```

In [62]:
```
mat = confusion_matrix(y_test, predicted)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[62]:
```
Text(91.68,0.5,'predicted label')
```

Out[62]:




In [63]:
```
# Instantiate
bayes_model = GaussianNB()
# Fit the model
bayes_model.fit(x_train, y_train)
# Accuracy
bayes_model.score(x_train, y_train)
```

Out[63]:
```
0.937375213918996
```

In [64]:
```
# Predictions/probs on the test dataset
predicted = pd.DataFrame(bayes_model.predict(x_test))
probs = pd.DataFrame(bayes_model.predict_proba(x_test))
```

In [65]:
```
# Store metrics
bayes_accuracy = metrics.accuracy_score(y_test, predicted)
bayes_roc_auc = metrics.roc_auc_score(y_test, probs[1])
```

```
        bayes_confus_matrix = metrics.confusion_matrix(y_test, predicted)
        bayes_classification_report = metrics.classification_report(y_test, predicted)
        bayes_precision = metrics.precision_score(y_test, predicted, pos_label=1)
        bayes_recall = metrics.recall_score(y_test, predicted, pos_label=1)
        bayes_f1 = metrics.f1_score(y_test, predicted, pos_label=1)
```

```
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/metrics/classification.py:114
  'precision', 'predicted', average, warn_for)
```

```
In [0]:
```

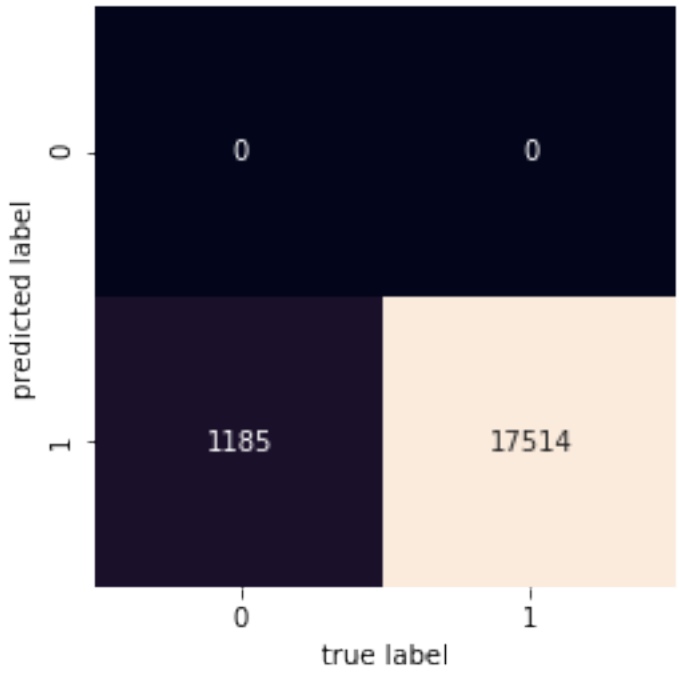
```
In [66]: print(bayes_confus_matrix)
```

```
[[    0  1185]
 [    0 17514]]
```

```
In [67]: mat = confusion_matrix(y_test, predicted)
         sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
         plt.xlabel('true label')
         plt.ylabel('predicted label')
```

```
Out[67]: Text(91.68,0.5,'predicted label')
```

```
Out[67]:
```



In [68]:
```
modeldf =pd.DataFrame([[tree_accuracy,rf_accuracy, svm_accuracy, knn_accuracy, bayes_a

df1 = pd.DataFrame(['accuracy', 'precision','recall','f1' ], columns=['measure'])


modeldf=df1.join(modeldf)



print(modeldf.dtypes)
```

```
measure            object
Decision Tree     float64
Random forest     float64
svc model         float64
knn neighbours    float64
Bayes             float64
dtype: object
```

In [69]:
```
modeldf
```

In [70]:
```
#More detail on Linear regression
# Create linear regression object
from sklearn.metrics import mean_squared_error, r2_score

regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)

y_pred = regr.predict(x_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
Coefficients:
 [ 3.50702794e-05  1.10638471e-03  1.56048103e-03 -4.58385482e-05
  3.16668012e-03 -1.65486455e-03 -1.87770721e-03  3.65891640e-04
  4.92375397e-05]
Mean squared error: 0.06
Variance score: -0.00
```

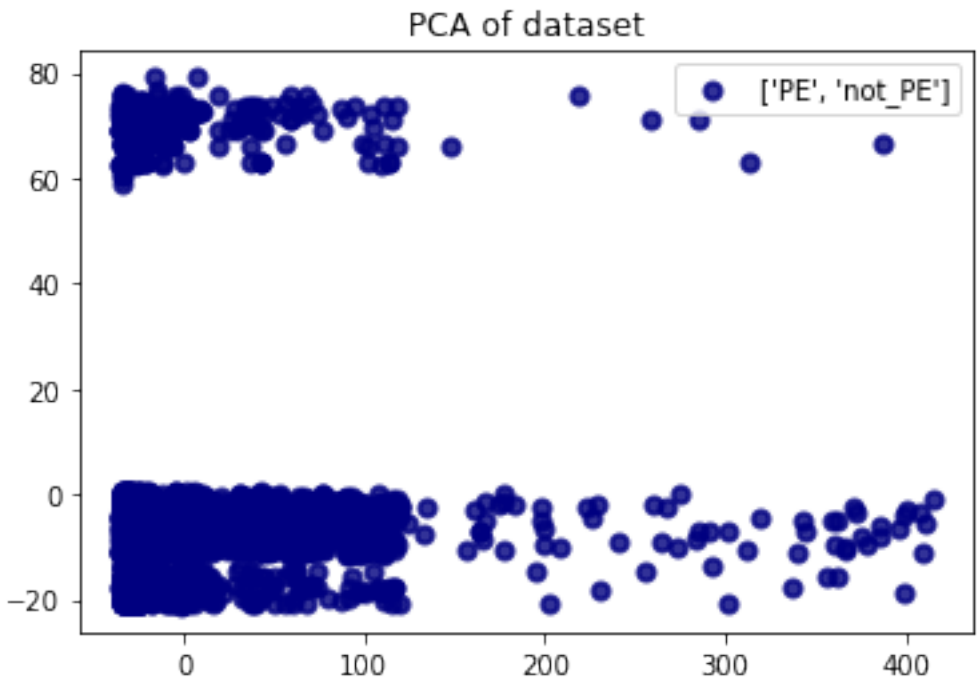
```
In [71]: from sklearn.decomposition import PCA
         pca = PCA(n_components=5)
         pca.fit(results)

Out[71]: PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)

In [87]: pca.components_

Out[87]: array([[ 9.99997895e-01,  4.12951078e-05, -9.93596840e-06,
                 -2.32763206e-04,  1.61016328e-06,  3.61856362e-05,
                 -1.95354900e-05, -1.82603095e-05, -2.03767698e-03],
                [ 1.44625456e-04, -3.71077081e-03, -1.04837411e-04,
                  9.99048124e-01,  2.35414390e-03,  3.04593079e-03,
                 -2.07394282e-03, -3.32613187e-03, -4.31145014e-02]])

In [88]: # Percentage of variance explained for each components
         print('explained variance ratio (first two components): %s'
               % str(pca.explained_variance_ratio_))

explained variance ratio (first two components): [0.8069189  0.11750461]


In [89]: X=x_train
         y=y_train


         feature_names=x_test.columns.tolist(),
         target_names=['PE', 'not_PE'],


         pca = PCA(n_components=2)
         X_r = pca.fit(X).transform(X)

         lda = LinearDiscriminantAnalysis(n_components=2)
         X_r2 = lda.fit(X, y).transform(X)

         # Percentage of variance explained for each components
         print('explained variance ratio (first two components): %s'
               % str(pca.explained_variance_ratio_))

         plt.figure()
         colors = ['navy', 'turquoise', 'darkorange']
         lw = 2

         for color, i, target_name in zip(colors, [0, 1, 2], target_names):
             plt.scatter(X_r[y == i, 0], X_r[y == i, 1], color=color, alpha=.8, lw=lw,
                         label=target_name)
         plt.legend(loc='best', shadow=False, scatterpoints=1)
```

```
        plt.title('PCA of dataset')

        plt.figure()
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):
            plt.scatter(X_r2[y == i, 0], X_r2[y == i, 1], alpha=.8, color=color,
                        label=target_name)
        plt.legend(loc='best', shadow=False, scatterpoints=1)
        plt.title('LDA of dataset')
```

```
explained variance ratio (first two components): [0.8069189  0.11750461]


/ext/sage/sage-8.6_1804/local/lib/python2.7/site-packages/sklearn/discriminant_analysis.py:388
  warnings.warn("Variables are collinear.")
```

```
        ---------------------------------------------------------------------------

        IndexError                                Traceback (most recent call last)

        <ipython-input-89-459ed32e0e5a> in <module>()
         29 plt.figure()
         30 for color, i, target_name in zip(colors, [Integer(0), Integer(1), Integer(2)], targ
    ---> 31     plt.scatter(X_r2[y == i, Integer(0)], X_r2[y == i, Integer(1)], alpha=RealNumbe
         32                 label=target_name)
         33 plt.legend(loc='best', shadow=False, scatterpoints=Integer(1))


        IndexError: index 1 is out of bounds for axis 1 with size 1
```

Out[89]:




Out[89]: <Figure size 432x288 with 0 Axes>

In [0]:

In [0]: