**Breast Cancer Wisconsin**

1. **Wstęp**

**Breast Cancer Wisconsin Dataset to baza danych zawierająca informacje o wynikach badań komórek guzów piersi oraz diagnozy czy dany nowotwór jest łagodny, czy złośliwy.**

**Baza danych zawiera 569 rekordów oraz 33 kolumny:**

1. **ID**
2. **Diagnosis - diagnoza nowotworu (M – malignant [złośliwy], B – benign [łagodny]**
3. **Radius mean – promień badanej komórki**
4. **Texture mean – struktura tkanki (wyrażona jako odchylenie standardowe skali szarości**
5. **Perimeter mean – obwód komórki**
6. **Area mean- obszar komórki**
7. **Smoothness mean**
8. **Compactness mean – ścisłość (perimeter ^2/area-1.0)**
9. **Concavity mean– wklęsłość**
10. **Concave points mean**
11. **Symmestry mean -**
12. **Fractal dimension mean**

Wartości 13-32 to wartości SE (standard error of the mean – wyznacznik dokładności średniej) oraz wartości worst (średnia z 3 najgorszych wyników) dla każdej z powyższych kategorii.

Do wykonania projektu wykorzystywałem paczki: numpy, pandas, pydotplus, matplotlib, sklearn, seaborn, IPython

1. **Opis eksperymentów**

Pracę nad bazą rozpocząłem od załadowania jej z pliku, sprawdzenia jej wielkości oraz występowania pustych rekordów.

data = pd.read\_csv('data.csv')  
print (data.shape)  
print data.isnull().sum()

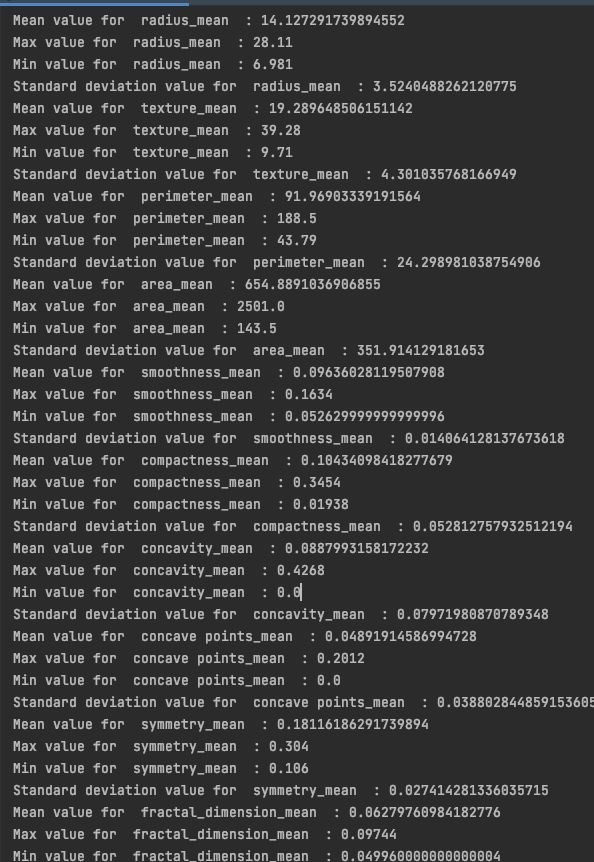
Baza danych zawierała kolumnę zawierającą wartości null dla każdego z rekordów. Do dalszej pracy usunąłem kolumnę ID oraz kolumnę z warościami null.

data.drop('Unnamed: 32', inplace=True, axis=1)  
data.drop('id', inplace=True, axis=1)

Wartości z kolumny „diagnosis”, zostały zmienione na wartości numeryczne – „M” = 1.0; „B” = 0.0.

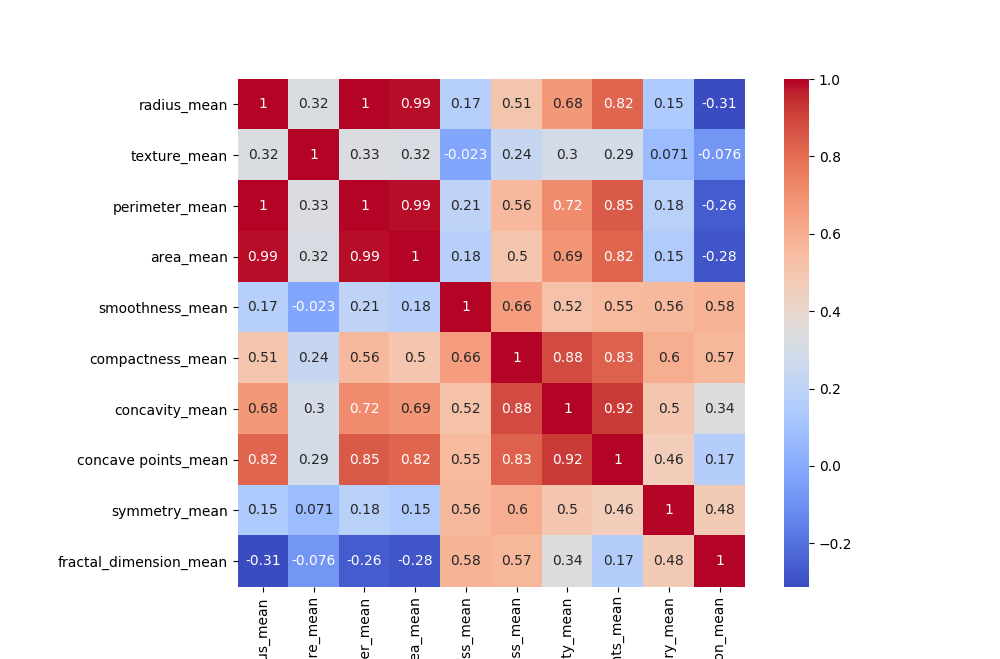
data['diagnosis'] = data["diagnosis"].replace({"M": 1.0, "B": 0.0})

Do wykonania zadania zdecydowałem się wybrać wartości średnie dla każdego z parametrów rekordu. Dla wybranych kolumn obliczyłem wartość średnią, standardowe odchylenie, wartość najmniejszą oraz wartość największą:



Następnie przygotowałem heatmapę – matrycę zależności pomiędzy poszczególnymi parametrami oraz scatter matrix – matrycę pokazującą dla każdego z paramentrów zależność pomiędzy wartością parametru. a wynikiem. Kolor czerwony oznacza guza złośliwego, a niebieski łagodnego.

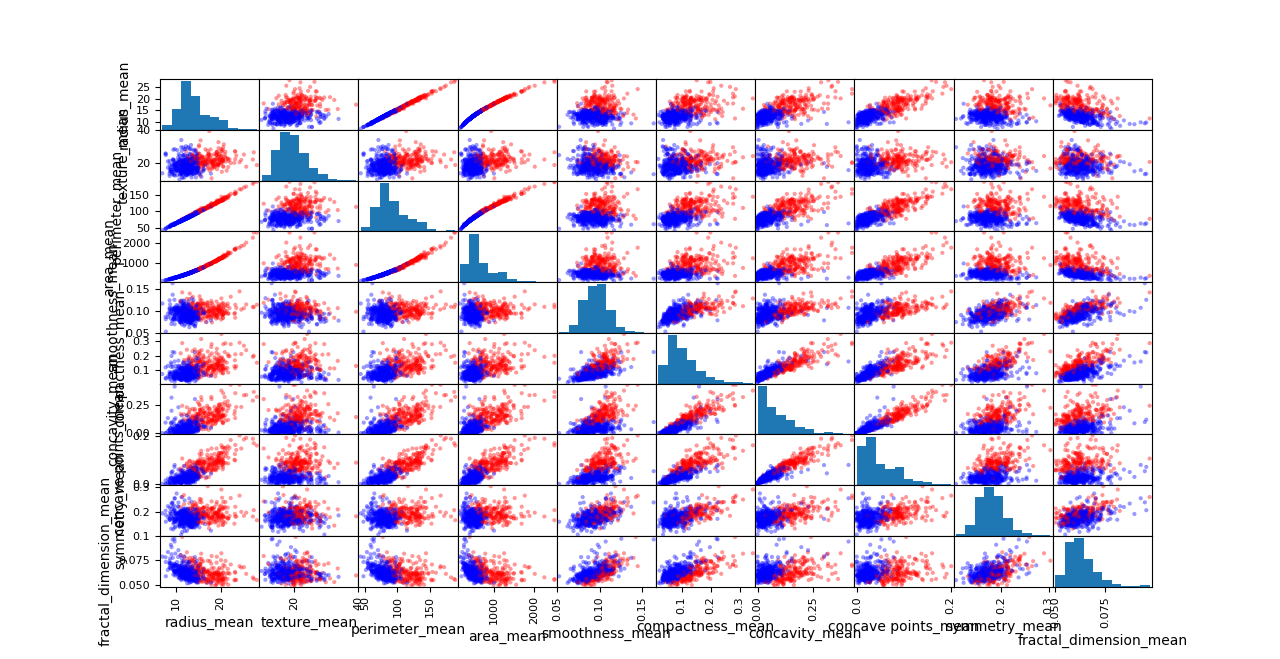
plt.figure(figsize=(10, 10))  
sns.heatmap(data[features\_mean].corr(), annot=True, square=True, cmap='coolwarm')  
color\_dic = {1.0: 'red', 0.0: 'blue'}  
colors = data['diagnosis'].map(lambda x: color\_dic.get(x))  
  
sm = scatter\_matrix(data[features\_mean], c=colors, alpha=0.4, figsize=(20, 20))

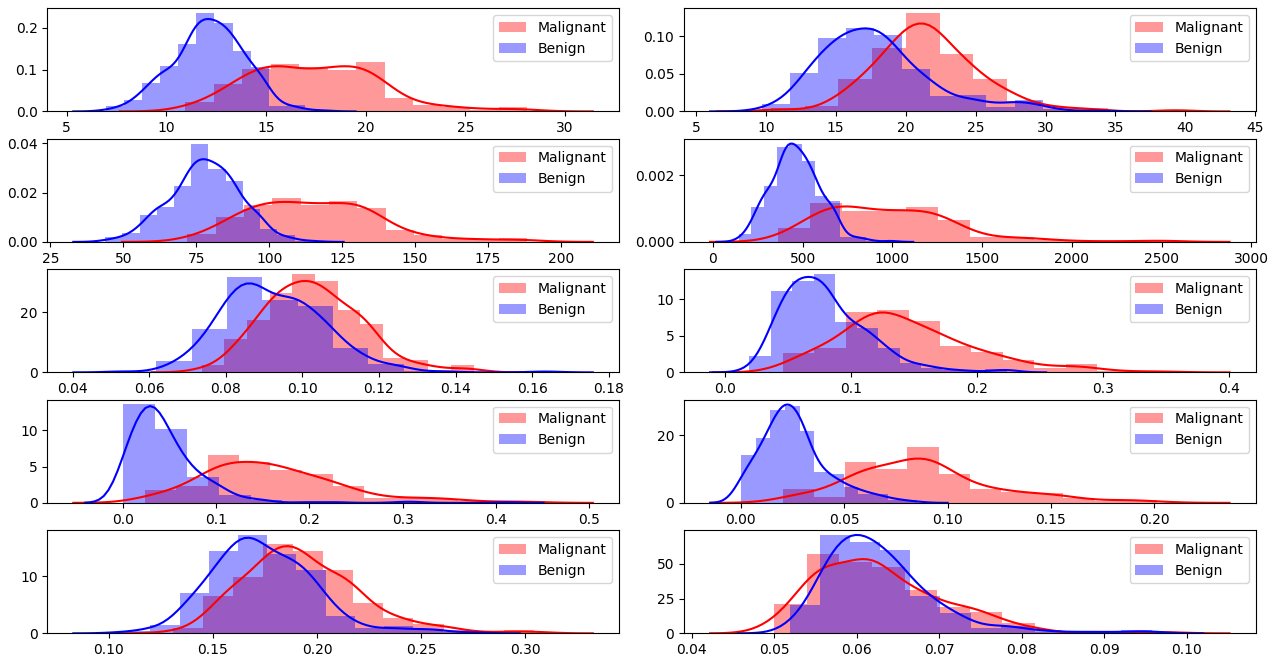


Kolejnym krokiem wizualizacji danych było przedstawienie jak zmiana wartości parametrów wpływa na diagnozę. Przedstawiłem to na dwóch wykresach:

for i, feature in enumerate(features\_mean):  
 rows = int(len(features\_mean) / 2)

plt.subplot(rows, 2, i + 1)  
 sns.distplot(data[data['diagnosis'] == 1.0][feature], bins=bins, color='red', label='Malignant');  
 sns.distplot(data[data['diagnosis'] == 0.0][feature], bins=bins, color='blue', label='Benign');  
  
 plt.legend(loc='upper right')  
  
lt.tight\_layout()  
plt.show()





1. **Klasyfikacja**

Do klasyfikacji wykorzystałem 8 algorytmów : drzewo decyzyjne, KNN, Stochastic Gradient Descent, Naive Bayes, Support Vector Machinec, Random Forest oraz sieć neuronową.

Klasyfikatory opierały się na wspólnym, wytrenowanym zbiorze.

Dla każdego klasyfikatora podana jest dokładność, sprawdzian krzyżowy (zbiór dzieli się na podzbiory, na których części wykonuje się analizę, a pozostałe są zbiorami kontrolnymi sprawdzającymi wiarygodność wyników) oraz czas wykonania. Dodatkowo, wynik każdego z klasyfikatorów przedstawiony jest za pomocą matrycy błędów.

start\_knn = time.time()  
  
[…]  
  
end\_knn = time.time()  
  
accuracy.append(accuracy\_score(prediction\_knn, y\_test))  
cvs.append(np.mean(scores\_knn))   
methods.append('KNN')  
  
print("Accuracy: {0:.2%}".format(accuracy\_score(prediction\_knn, y\_test)))  
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(np.mean(scores\_knn), np.std(scores\_knn) \* 2))  
print("Execution time: %s seconds \n" % "{0:.5}".format(end\_knn - start\_knn))

Podział i przygotowanie zbioru:

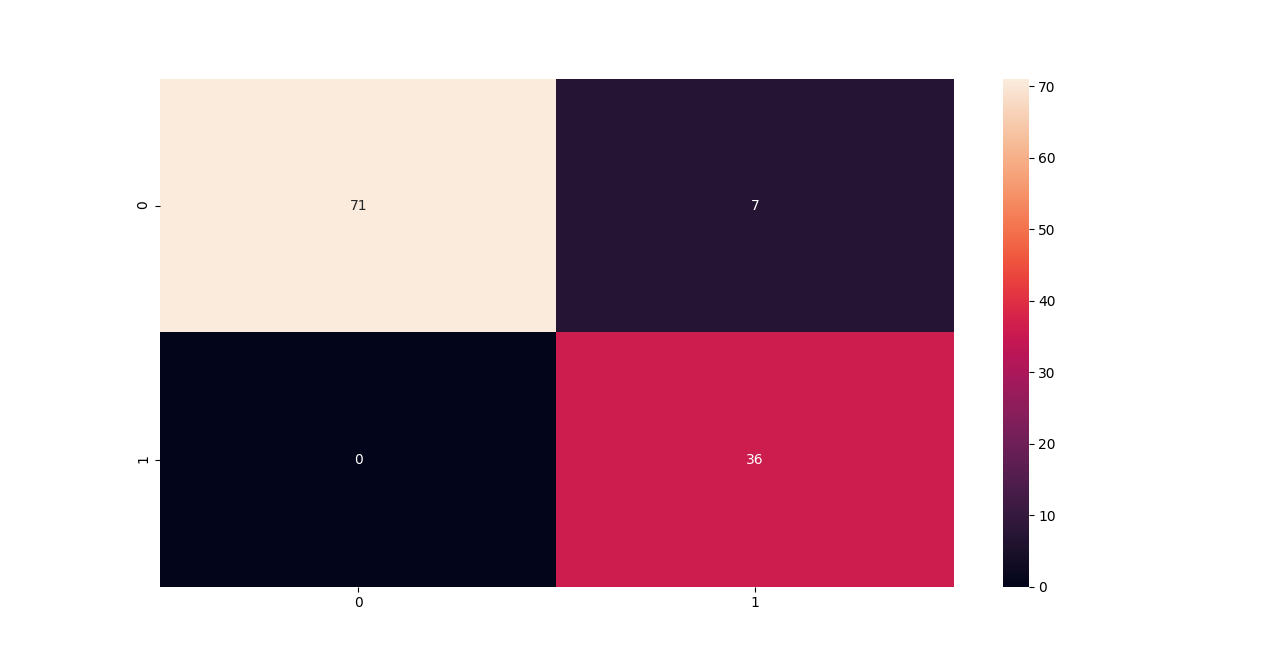
X = data.loc[:, features\_mean]  
y = data.loc[:, 'diagnosis']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

accuracy = []  
cvs = []  
methods = []

1. K Nearest Neighbors

knn = KNeighborsClassifier(n\_neighbors=10)   
knn.fit(X\_train, y\_train)  
prediction\_knn = knn.predict(X\_test)  
scores\_knn = cross\_val\_score(knn, X, y, cv=5)

|  |  |
| --- | --- |
| KNN | |
| Accuracy | **93.86%** |
| Cross validation scrore | **89.13% (+/- 8.34%)** |
| Execution time | **0.061685 seconds** |



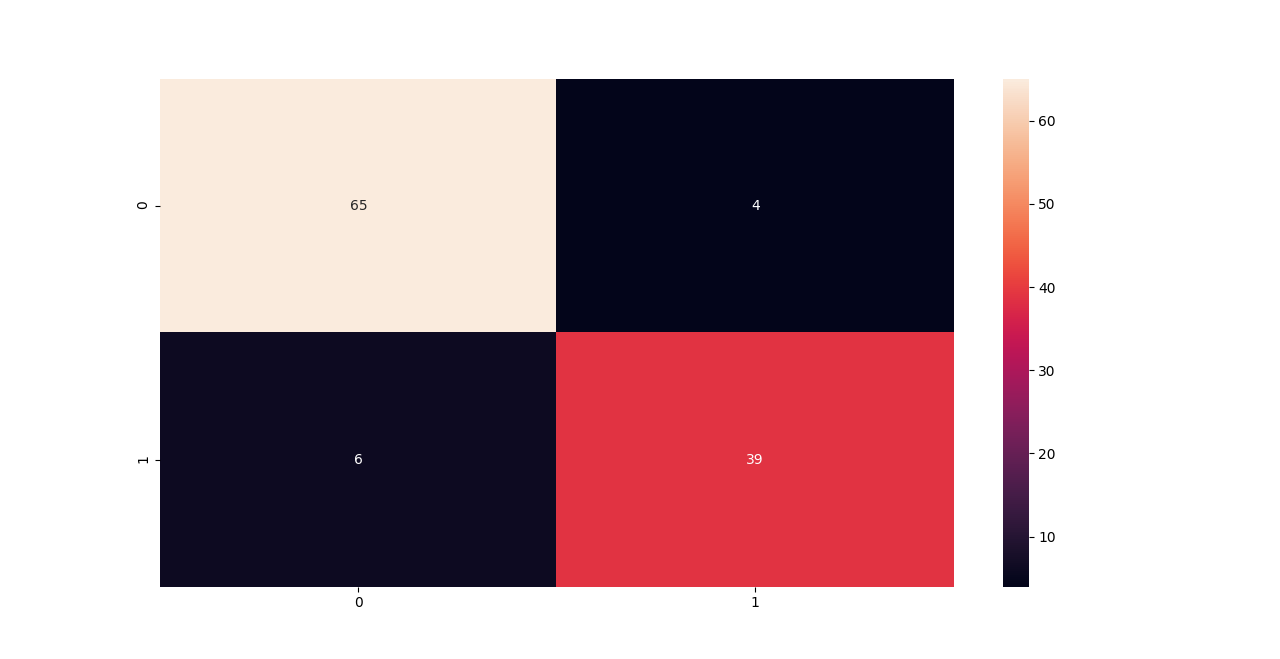
1. Drzewo decyzyjne

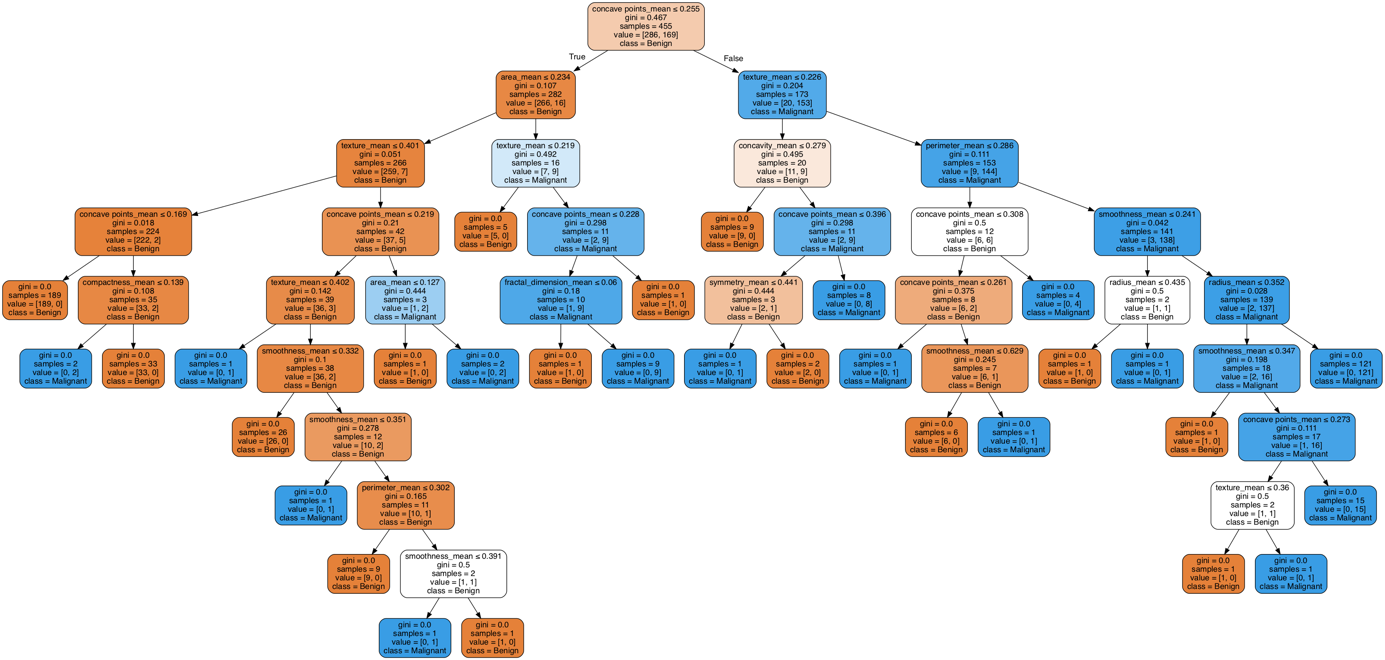
start\_dt = time.time()  
  
dt = DecisionTreeClassifier()  
dt.fit(X\_train, y\_train)  
prediction\_dt = dt.predict(X\_test)  
score = dt.score(X\_test, y\_test)  
scores\_dt = cross\_val\_score(dt, X, y, cv=5)  
  
end\_dt = time.time()  
  
accuracy.append(accuracy\_score(prediction\_dt, y\_test))  
cvs.append(np.mean(scores\_dt))  
methods.append('Decision Tree')

Zapisanie wytrenowanego drzewa decyzyjnego do pliku:

dot\_data = StringIO()  
export\_graphviz(dt, out\_file=dot\_data,  
 filled=True, rounded=True,  
 special\_characters=True, feature\_names=list(X), class\_names=['Benign', 'Malignant'])  
graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())  
graph.write\_png('breast\_cancer\_wisconsin.png')  
Image(graph.create\_png())

|  |  |
| --- | --- |
| Decision Tree | |
| Accuracy | **91.23%** |
| Cross validation scrore | **92.27% (+/- 3.00%)** |
| Execution time | **0.039065 seconds** |

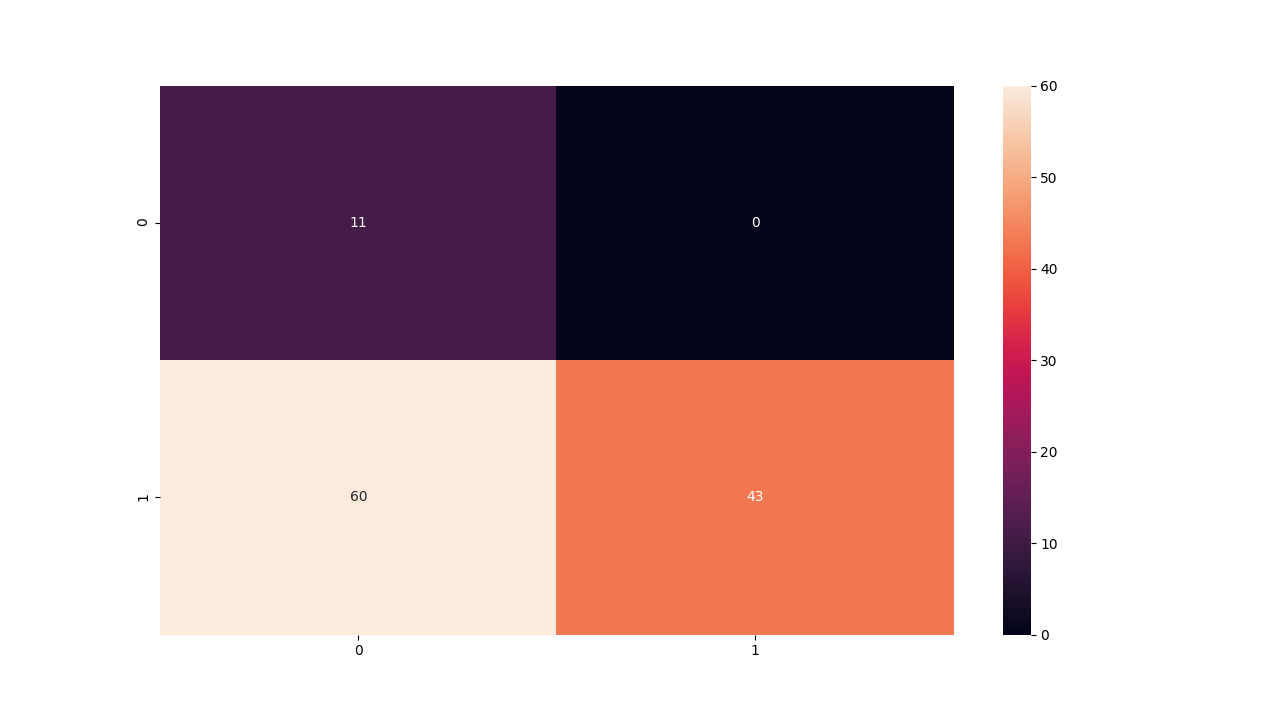




1. Stochastic Gadient Descent

start\_sgd = time.time()  
  
sgd = SGDClassifier()  
sgd.fit(X\_train, y\_train)  
prediction\_sgd = sgd.predict(X\_test)  
scores\_sgd = cross\_val\_score(sgd, X, y, cv=5)  
  
end\_sgd = time.time()  
  
accuracy.append(accuracy\_score(prediction\_sgd, y\_test))  
cvs.append(np.mean(scores\_sgd))  
methods.append('SGD')

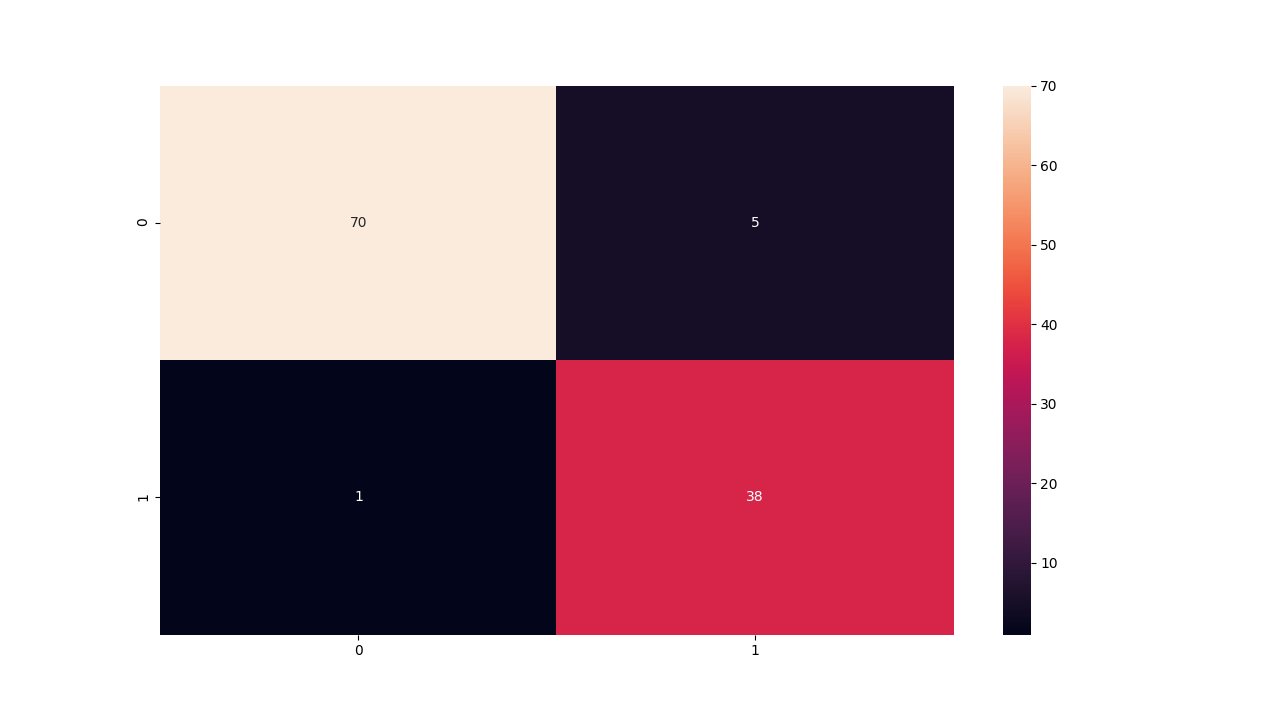
|  |  |
| --- | --- |
| Stochastic Gadient Descent | |
| Accuracy | **47.37%** |
| Cross validation scrore | **85.96% (+/- 5.67%)** |
| Execution time | **0.039719 seconds** |



1. Naïve Bayes

start\_nb = time.time()  
  
nb = GaussianNB()  
nb.fit(X\_train, y\_train)  
prediction\_nb = nb.predict(X\_test)  
scores\_nb = cross\_val\_score(nb, X, y, cv=5)  
  
end\_nb = time.time()  
  
accuracy.append(accuracy\_score(prediction\_nb, y\_test))  
cvs.append(np.mean(scores\_nb))  
methods.append('Naive Bayes')  
  
sns.heatmap(confusion\_matrix(prediction\_nb, y\_test), annot=True, fmt="d")

|  |  |
| --- | --- |
| Naïve Bayes | |
| Accuracy | **94.74%** |
| Cross validation scrore | **91.40% (+/- 5.03%)** |
| Execution time | **0.026034 seconds** |

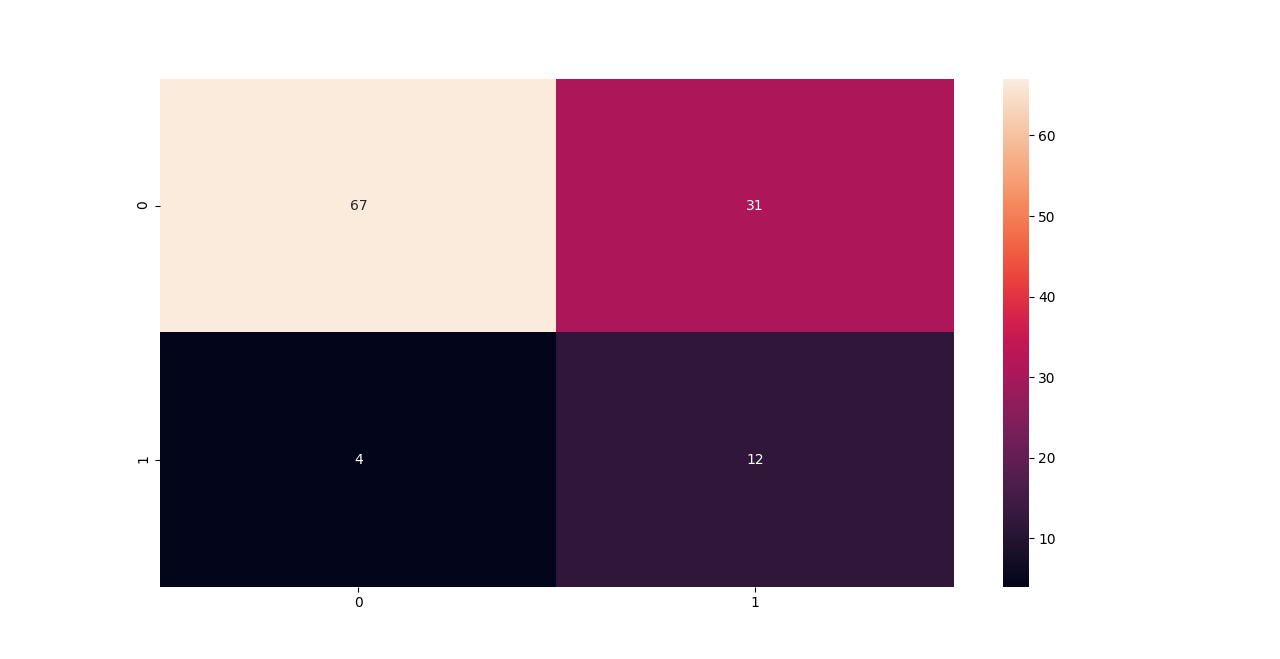


1. Support Vector Machines

SVC:

svc = SVC()  
svc.fit(X\_train, y\_train)  
prediction\_svc = svc.predict(X\_test)  
scores\_svc = cross\_val\_score(svc, X, y, cv=5)  
  
end\_svc = time.time()  
  
accuracy.append(accuracy\_score(prediction\_svc, y\_test))  
cvs.append(np.mean(scores\_svc))  
methods.append('SVC')  
  
sns.heatmap(confusion\_matrix(prediction\_svc, y\_test), annot=True, fmt="d")

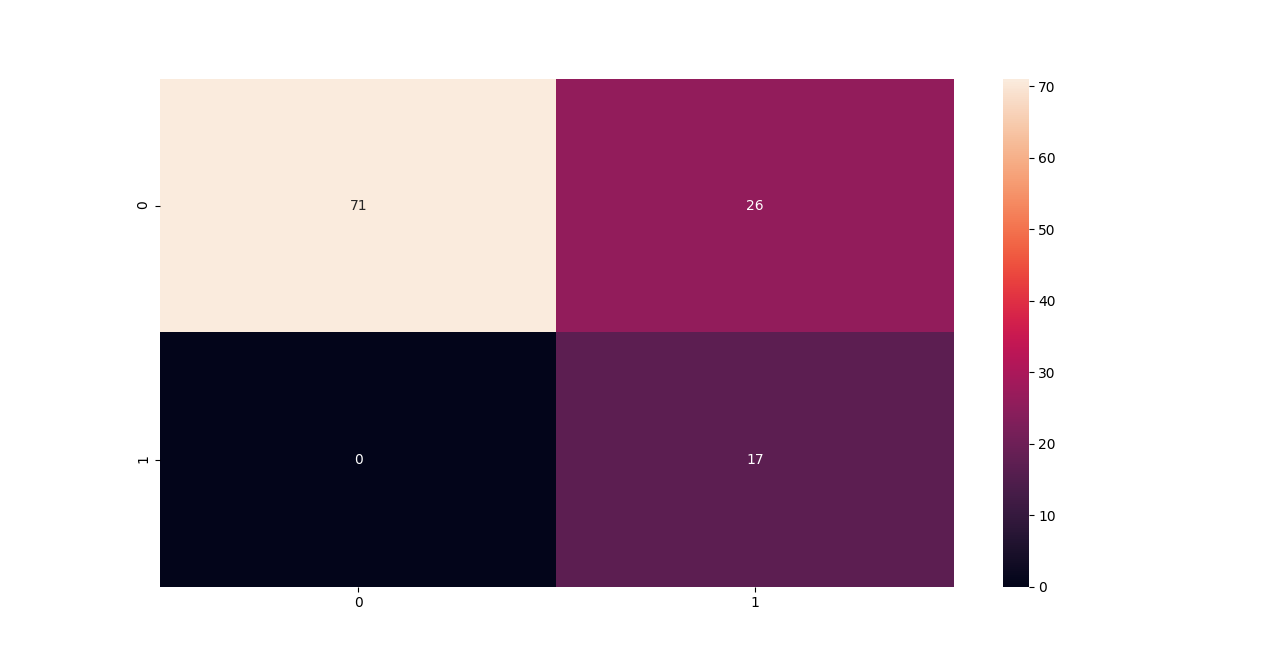
|  |  |
| --- | --- |
| SVC | |
| Accuracy | **69.30%** |
| Cross validation scrore | **71.70% (+/- 4.07%)** |
| Execution time | **0.11236 seconds** |



Linear SVC:

start\_Linearsvc = time.time()  
  
LinearSVC = LinearSVC()  
LinearSVC.fit(X\_train, y\_train)  
prediction\_LinearSVC = LinearSVC.predict(X\_test)  
scores\_LinearSVC = cross\_val\_score(LinearSVC, X, y, cv=5)  
  
end\_LinearSVC = time.time()  
  
accuracy.append(accuracy\_score(prediction\_LinearSVC, y\_test))  
cvs.append(np.mean(scores\_LinearSVC))  
methods.append('Linear SVC')

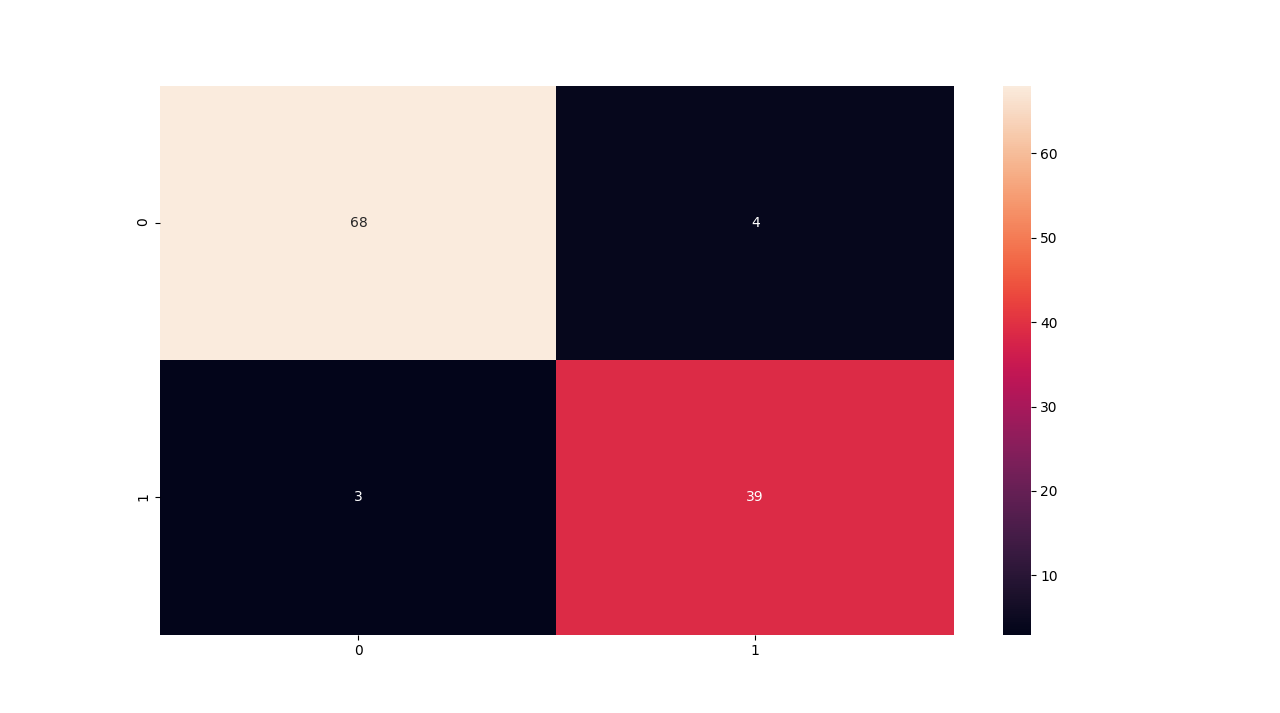
|  |  |
| --- | --- |
| Linear SVC | |
| Accuracy | **77.19%** |
| Cross validation scrore | **83.87% (+/- 10.89%)** |
| Execution time | **0.19297 seconds** |



1. Random Forest

start\_rf = time.time()  
  
rf = RandomForestClassifier()  
rf.fit(X\_train, y\_train)  
prediction\_rf = rf.predict(X\_test)  
scores\_rf = cross\_val\_score(rf, X, y, cv=5)  
  
end\_rf = time.time()  
  
accuracy.append(accuracy\_score(prediction\_rf, y\_test))  
cvs.append(np.mean(scores\_rf))  
methods.append('RF')  
  
# sns.heatmap(confusion\_matrix(prediction\_rf, y\_test), annot=True, fmt="d")

|  |  |
| --- | --- |
| Random Forest | |
| Accuracy | **93.86%** |
| Cross validation scrore | **93.69% (+/- 5.38%)** |
| Execution time | **0.25935 seconds** |



1. Sieci neuronowe

Dla sieci neuronowych, przed przystąpieniem do klasyfikacji zbiór został zmormalizowany.

def remove\_outliers(dataframe, column\_name):  
 print column\_name  
 values = dataframe[column\_name]  
 q1, q3 = np.percentile(values, [25, 75])  
 iqr = q3 - q1  
 median = values.median()  
 low = median - 1.5 \* iqr  
 high = median + 1.5 \* iqr  
 dataframe\_no\_outliers = dataframe[(dataframe[column\_name] >= low) & (dataframe[column\_name] <= high)]  
 return dataframe\_no\_outliers  
  
  
for column in data.drop(columns='diagnosis'):  
 data\_temp = remove\_outliers(data, column)

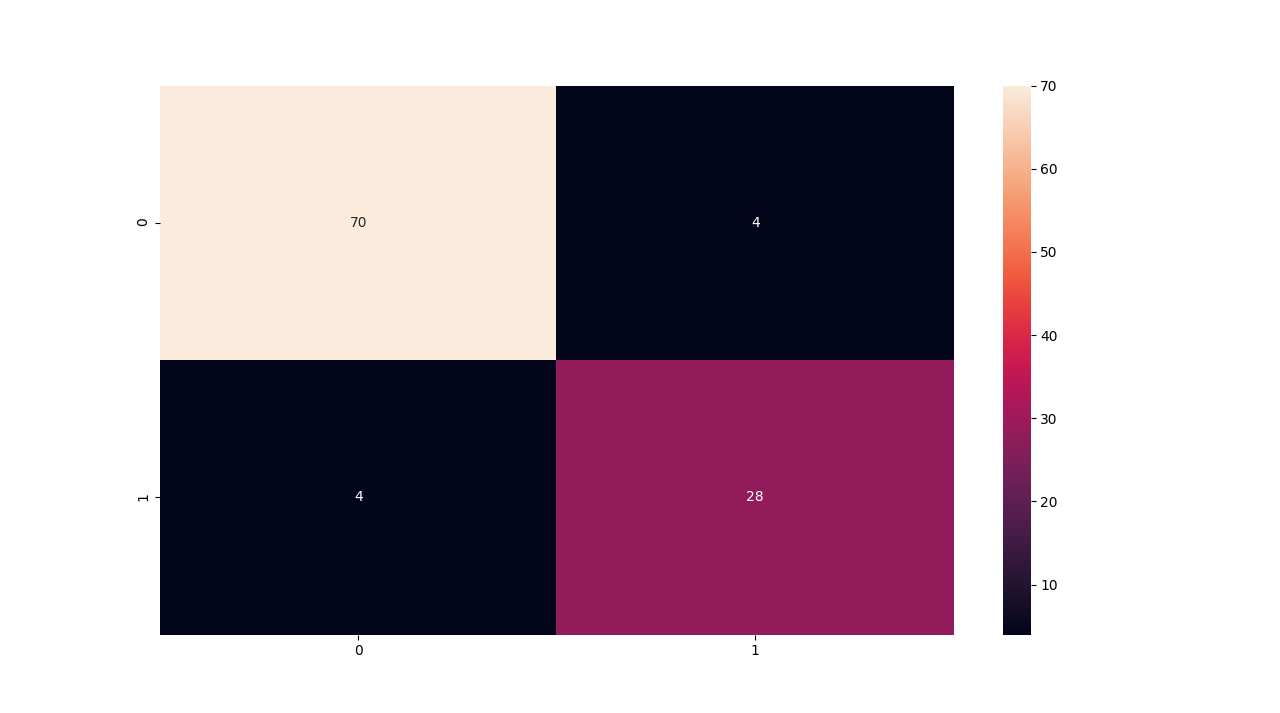
dataframe\_values = data\_temp.values  
min\_max\_scaler = preprocessing.MinMaxScaler()  
x\_scaled = min\_max\_scaler.fit\_transform(data\_temp)  
normalized\_dataframe = pd.DataFrame(x\_scaled)  
X = data\_temp.drop(columns='diagnosis', axis=1)  
y = data\_temp['diagnosis']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

Usunięcie wartości odstających oraz wytrenowanie znormalizowanego zbioru.

start\_MLPC = time.time()  
  
classifier = MLPClassifier(hidden\_layer\_sizes=(6, 3), max\_iter=500, activation='relu')  
model = classifier.fit(X\_train, y\_train)  
  
prediction\_MLPC = classifier.predict(X\_test)  
scores\_MLPC = cross\_val\_score(classifier, X, y, cv=5)  
  
end\_MLPC = time.time()  
  
accuracy.append(accuracy\_score(prediction\_MLPC, y\_test))  
cvs.append(np.mean(scores\_MLPC))  
methods.append('MLPC')

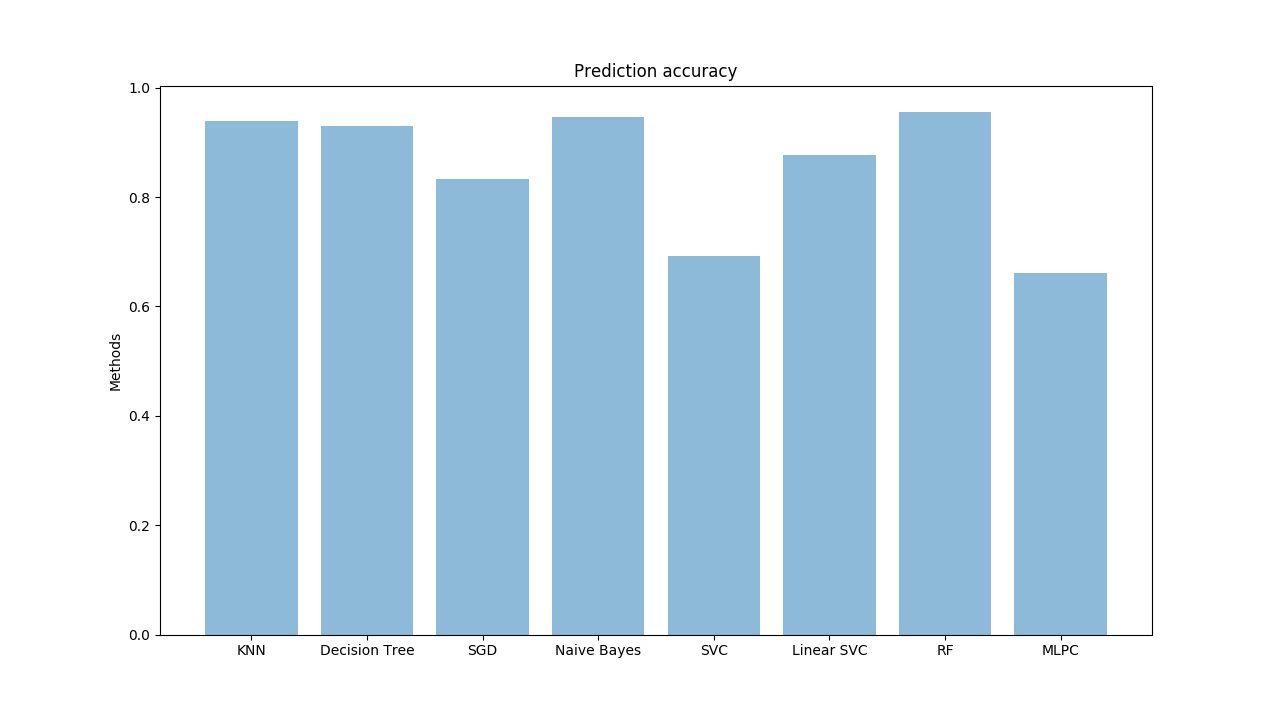
Klasyfikacja

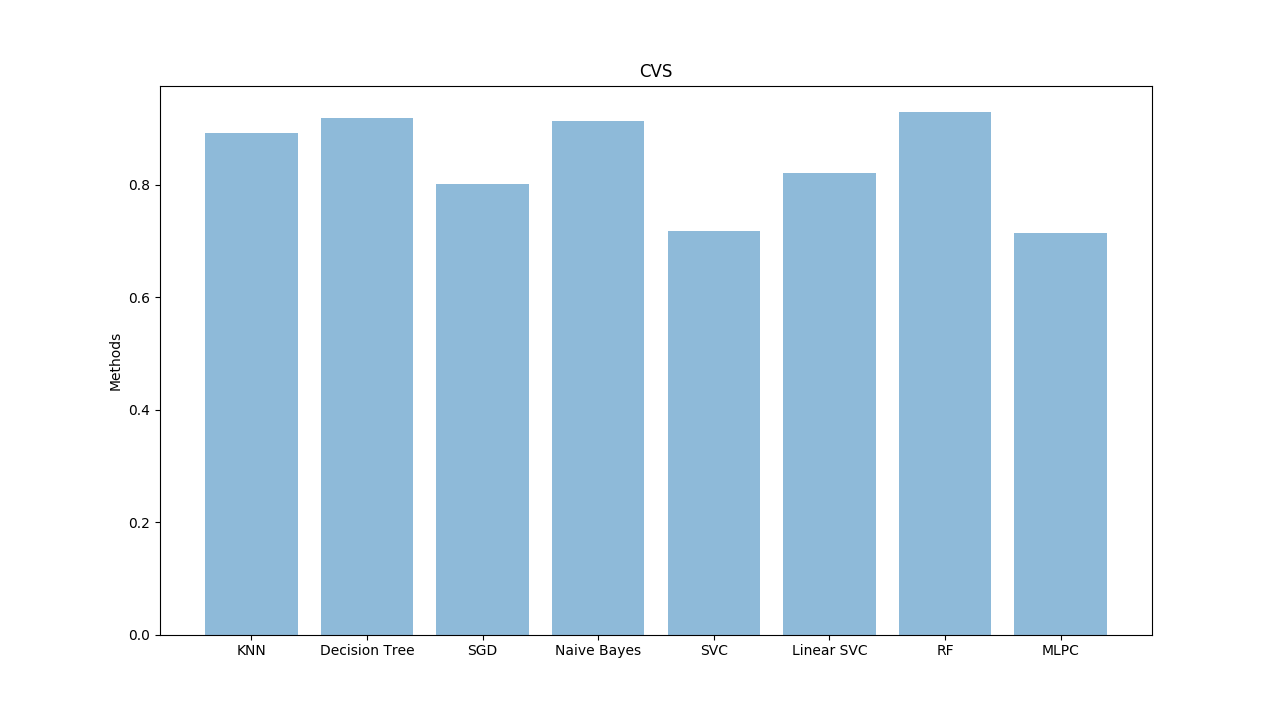
|  |  |
| --- | --- |
| Sieć neuronowa | |
| Accuracy | **68.87%** |
| Cross validation scrore | **73.92% (+/- 20.84%)** |
| Execution time | **2.1802 seconds** |



**4) Podsumowanie**

Poniższe wykresy przedstawiają dokładności oraz sprawdzian krzyżowy każdego z wykorzystanych klasyfikatorów





5) Bibliografia

1. Wykłady oraz ćwiczenia laboratoryjne
2. <https://www.kaggle.com/uciml/breast-cancer-wisconsin-data>
3. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29