Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проектна задача по предметот

Вовед во препознавање на облици

Детекција на рак на дојка со помош на модели изработени со машинско учење

(Breast Cancer Detection Using Machine Learning Models)

Изработил:

Мартин Штерјоски 151070

Професор:

Д-р Дејан Ѓорѓевиќ

Август 2019 година

Содржина

Вовед

Во овој проект опфатена е постапката за креирање на сопствена пајтон програма за откривање на карцином на дојка со помош на машинско учење врз база на податоци. Ракот на дојка е еден од најчестите карциноми кај жените ширум светот, што претставува најголем број нови случаи на рак и смртни случаи поврзани со рак според глобалната статистика, што го прави значаен проблем на јавното здравство во денешното општество.

Раната дијагностика на ракот на дојка може значително да ја подобри прогнозата и шансата за преживување, бидејќи може да промовира навремено клиничко лекување кај пациентите. Понатамошна точна класификација на бенигни тумори може да ги спречи пациентите да поминат непотребни третмани. Така, правилното дијагностицирање на ракот на дојка и класификација на пациентите во малигни или бенигни групи е предмет на многу истражувања. Поради неговите уникатни предности при откривање на критични карактеристики од комплексни податоци за ракот на дојка, машинското учење (МЛ) е широко признато како методологија за класификацијата на моделите.

Најважниот тест за скрининг за рак на дојка е мамограмот. Мамограм е рендген на градите. Може да открие карцином на дојка до две години пред да се почувствува туморот од вас или вашиот лекар.

Подготовка на податоците

Користена е базата на податоци од https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. Карактеристиките се пресметуваат од дигитализирана слика на тенка игла аспират (FNA - fine needle aspirate) на маса на дојка. Тие ги опишуваат карактеристиките на клеточните јадра, присутни на сликата.

Информации за атрибут:

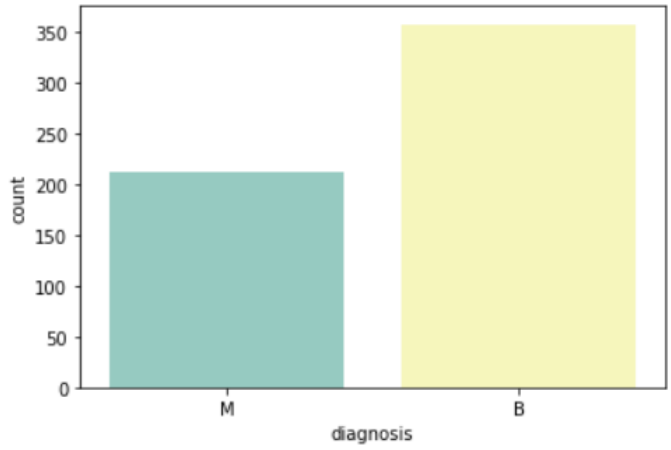
1) матичен број 2) Дијагноза (М = малигни, Б = бенигна) 3-32)

Десет реални вредности се пресметуваат за секое клеточно јадро:

а) радиус (средна оддалеченост од центар до точки на периметарот)
б) текстура (стандардна девијација на вредности во сива скала)
в) периметар
г) област
д) мазност (локална варијација во должината на радиусот)
ѓ) компактност (периметар ^ 2 / област - 1,0)
е) конкавност (сериозност на конкавни делови од контурата)
ж) конкавни точки (број на конкавни делови на контурата)
з) симетрија
s) фрактална димензија ("приближување на крајбрежјето" - 1)

Средната, стандардната грешка и „најлошата“ или најголемата (просечно од трите најголеми вредности) на овие карактеристики се пресметани за секоја слика, што резултира со 30 карактеристики. Сите вредности на карактеристиките се кодираат со четири значајни цифри.

Дистрибуција на класи: 357 бенигни, 212 малигни



Изглед на табелата со податоци (неколку колони):

	# id ID number	✔ diagnosis The diagnosis of breast tissues (M = malignant, B = benign)	# radius_mean mean of distances from center to points on the perimeter	# texture_mean standard deviation of gray-scale values	# perimeter_mean mean size of the core tumor	# area_mean	# smoothness_mean mean of local variation in radius lengths
1	842302	M	17.99	10.38	122.8	1001	0.1184
2	842517	M	20.57	17.77	132.9	1326	0.08474
3	84300903	M	19.69	21.25	130	1203	0.1096
4	84348301	M	11.42	20.38	77.58	386.1	0.1425
5	84358402	M	20.29	14.34	135.1	1297	0.1003
6	843786	M	12.45	15.7	82.57	477.1	0.1278
7	844359	M	18.25	19.98	119.6	1040	0.09463
8	84458202	M	13.71	20.83	90.2	577.9	0.1189
9	844981	M	13	21.82	87.5	519.8	0.1273
10	84501001	M	12.46	24.04	83.97	475.9	0.1186

Сумаризација на податоците:

```
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: 144.5+ KB
```

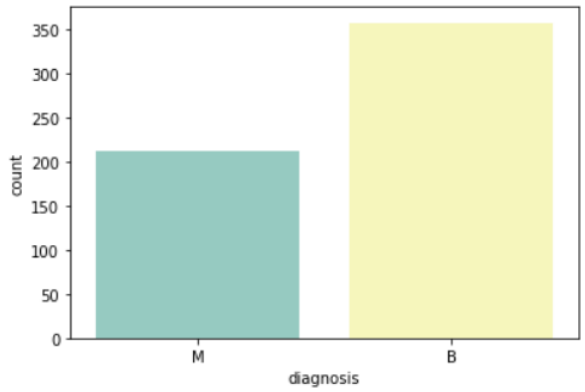
Визуелизација на податоците

Со користење на различни алатки за визуелизацијата на податоци може полесно да се разберат и исто така да се објаснат податоците за овој проект. Пајтон има неколку интересни библиотеки за визуелизација, како што се Matplotlib, Seaborn итн. Со употреба на овие алатки исцртани се различни дијаграми за визуелен приказ на различни типови на податоци како на пример:

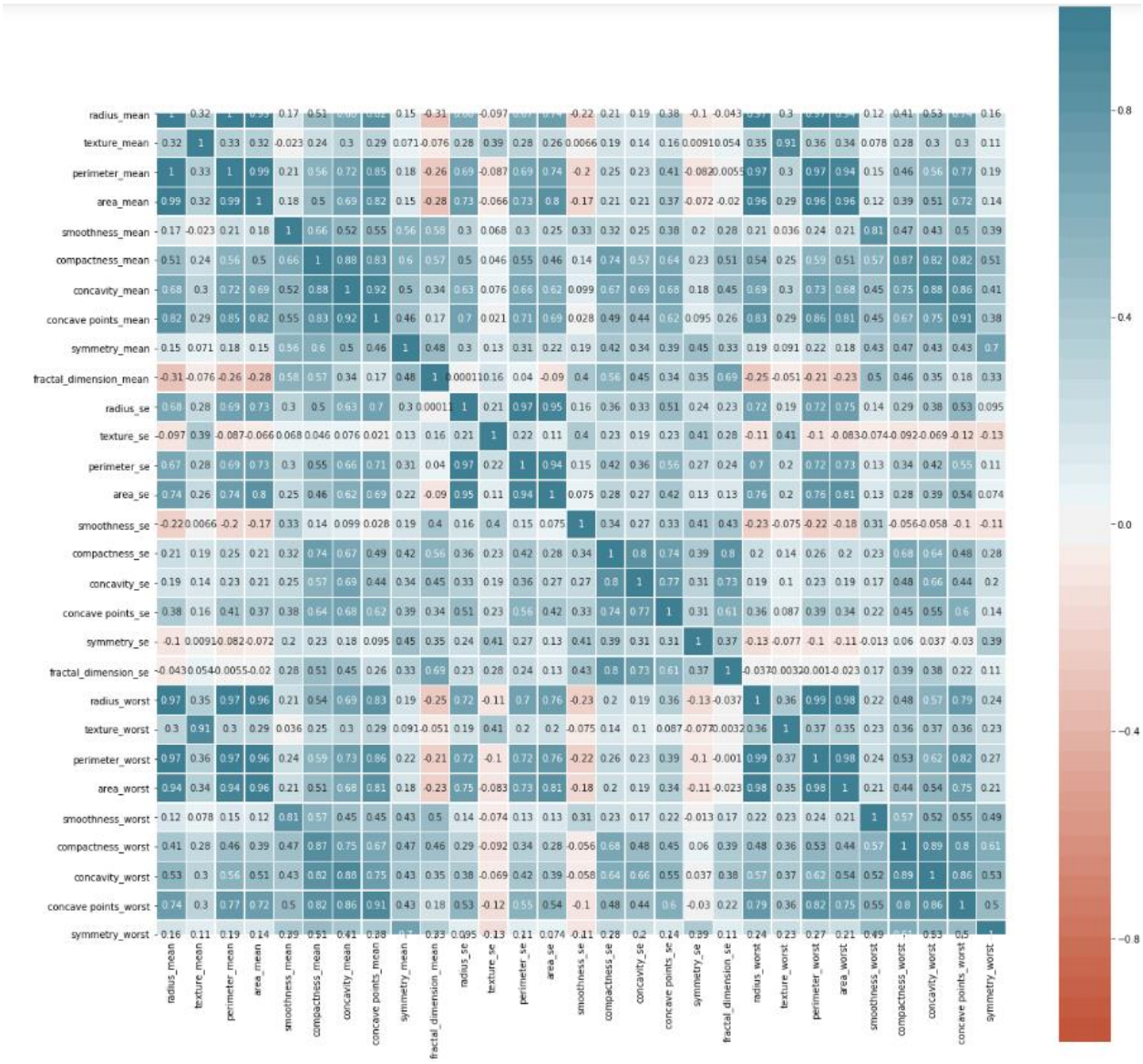
1. **countplot** фунцкијата од **seaborn** за приказ на вкупен број на двете дијагнози. На вертикалната оска го имаме бројот на дијагноза, а на хоризонталната оска ја имаме дијагнозата.

```
# Visualize this count
sns.countplot(dataSet['diagnosis'], label="Count", palette="Set3")
```

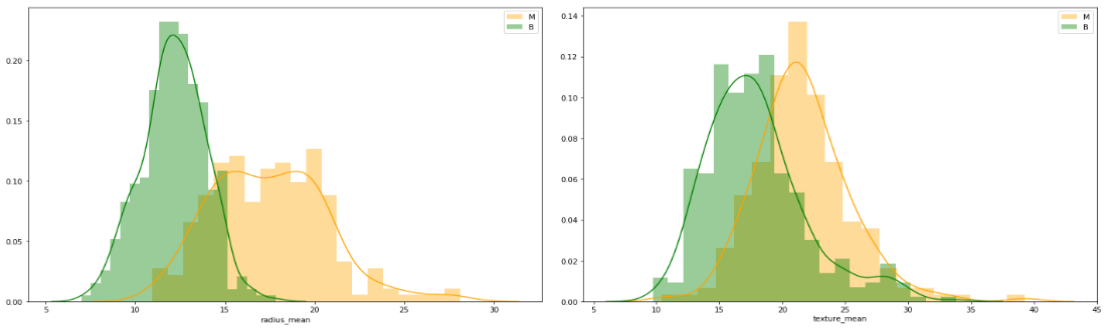
```
<matplotlib.axes._subplots.AxesSubplot at 0xcefc850>
```



2. **heatmap** функцијата од **seaborn** за исцртување на мапа за приказ на корелацијата помеѓу податоците. Темно плавата боја значи дека има позитивна корелација помеѓу податоците, а темно црвената боја означува негативна корелација на податоците. Кога пак корелацијата е нула или блиску нулата, тогаш бојата е сива (слика). Na пример, „perimeter_worst“ и „texture_mean“ се многу поврзани, а „smoothness_woorst“ и „area_mean“ има корелација од околу 0,57. Така, со овој мапа (heatmap) на корелациите помеѓу карактеристиките ќе добиеме подобра идеја за тоа како можеме да донесеме одлуки за следните чекори на подготовка и анализа на податоците.

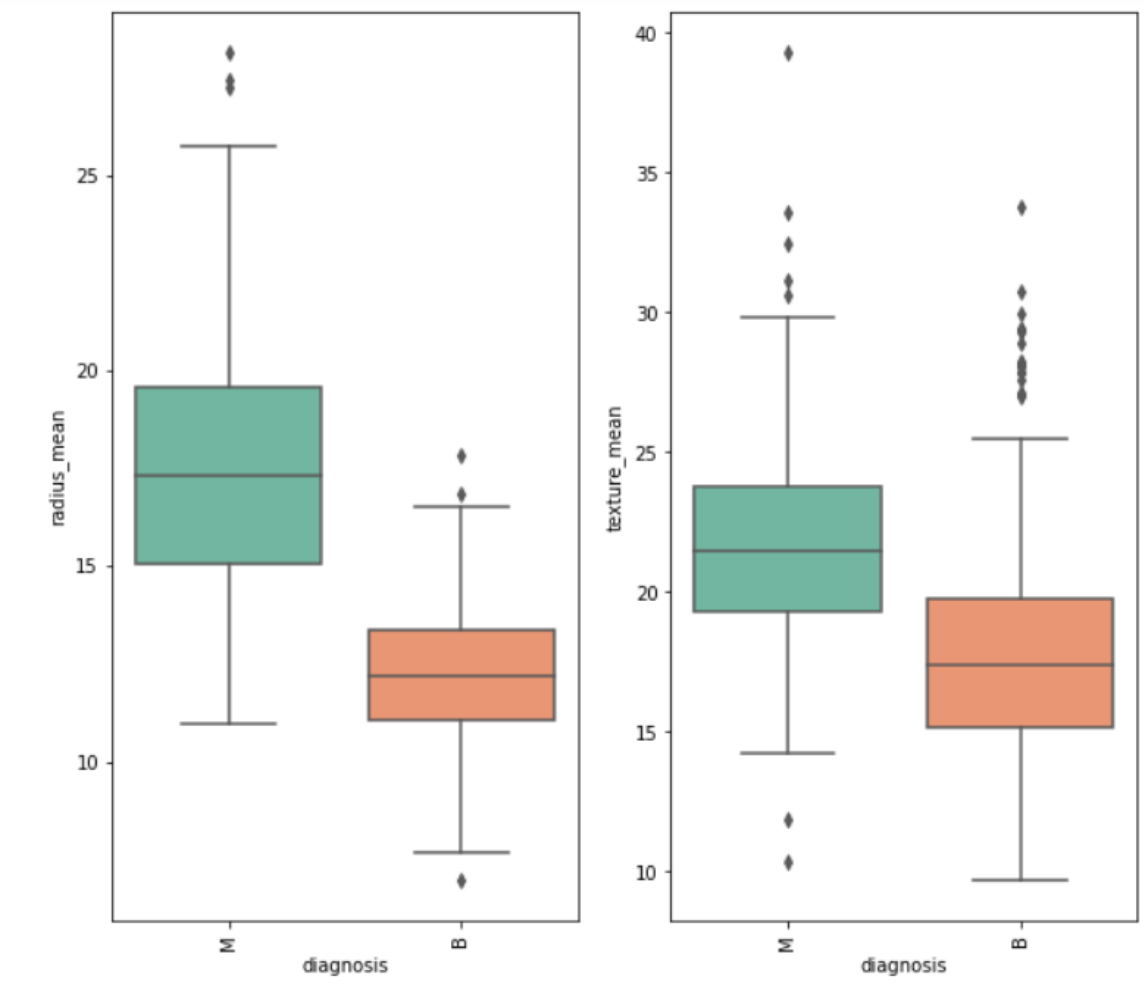


3. **distplot** функцијата од **seaborn** за исцртување на графици со која може да се види како малигните или бенигните клетки на туморите можат да имаат (или не) различни вредности за карактеристиките што ја исцртуваат дистрибуцијата на секој вид дијагноза за секоја од средните карактеристики.



Користени се 20 бинови (колку поголем број на бинови толку подобар резултат).

4. **boxplot** функцијата од **seaborn** која е искористена за претставување на дистрибуцијата на податоците базирана на пет точки (минимум, прв квартал, медијана, трет квартал и максимум). Исто така со овај график можеме да ги забележиме податоците кои се ‘outliers.’
   Во сликата подолу можеме да го споредиме опсегот на ‘radius_mean’ и ‘texture_mean’ за малигни и бенигни дијагнози. Од сликата може да се заклучи со голема самодоверба дека вистинските медијани се разликуваат бидејќи коцките не се преклопуваат.



Употреба на алгоритмите од машинско учење и поединечни резултати

Во делот со користење на алгоритмите од машинско учење опфатени се повеќе класификатори. Во ова податочно множество имаме зависна променлива, т.е. има само две групи на вредности, или М (малигна) или Б (бенигна). Значи, ние ќе го користиме алгоритмот за класификација на надгледуваното учење (учење во кое се обезбедени влезни и посакувани излезни податоци. Влезните и излезните податоци се означени за класификација, за да обезбедат основа за учење за идна обработка на податоците. Проблемите со надгледувано учење можат дополнително да се групираат во проблеми со регресија и класификација).

Алатки кои се користени во проектот во делот со пресметки и алгоритми се од пакетот **Scikit-Learn** .

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

Бидејќи алгоритмите кои ќе ги користиме во проектот работат со нумерички вредности ние соодветно ќе ги мапираме категориите на *M* и *B* во *1* и *0*.

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

Се со цел да се избегне ***Overfiting*** на податоците, ќе ја користиме функцијата **train_test_split** за рандом поделба на податочното множество на тест множество (кое ќе биде 20% од вкупното множество) и тренинг множество (80% од вкупното множество на податоци).

```
X = dataSet.loc[:, features_mean] # locating all the rows for the 'features_mean' column
y = dataSet.loc[:, 'diagnosis']   # locating all the rows for the 'diagnosis' column


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 50)
```

Во оваа функција атрибутот **random_state** е поставен на 50. Овај параметар претставува семе од кое се генерира рандом број.

За пресметување на резултатот на сите спроведени експерименти (Accuracy, Precision, Recall и F1 Score), направена е функцијата **report(classifier, classifierName)** која како аргументи ги прима соодветниот класификатор и неговото име.

```
def report(classifier, classifierName):

    Y_pred = classifier.predict(X_test)
    cm = confusion_matrix(y_test, Y_pred)

    print('Classification report for the : {}'.format(classifierName))
    print("Confusion Matrix: ")
    print(cm)

    TN = cm[0][0]
    TP = cm[1][1]
    FN = cm[1][0]
    FP = cm[0][1]

    # accuracy: (tp + tn) / (p + n)
    accuracy = accuracy_score(Y_pred, y_test)
    print('Accuracy: {0:.2%}'.format(accuracy))

    # precision tp / (tp + fp)
    precision = precision_score(Y_pred, y_test)
    print('Precision: {0:.2%}'.format(precision))

    # recall: tp / (tp + fn)
    recall = recall_score(Y_pred, y_test)
    print('Recall: {0:.2%}'.format(recall))
    #print('{} Recall: {}'.format(classifierName,  (TP / (TP + FN ))))

    # f1: 2 tp / (2 tp + fp + fn)
    f1 = f1_score(Y_pred, y_test)
    print('F1 score: {0:.2%}'.format(f1))
```

Спроведување на експериментите и приказ на нивните резултати од класификацијата:

1. **Stochastic Gradient Descent classifier** – Овој алгоритам е повикан во циклус со различни вредности за крос-валидацијата.

```
# Stochastic Gradient Descent classifier
from sklearn.linear_model import SGDClassifier
cvScore = 2

def sgdClf(cvScore):
    print("Stochastic Gradient Descent Classifier with cross validation:", cvScore)

    start = time.time()

    classifier = SGDClassifier()
    classifierName = "Stochastic Gradient Descent Classifier";
    classifier.fit(X_train, y_train) # fitting

    prediction = classifier.predict(X_test) # prediction
    scores     = cross_val_score(classifier, X, y, cv=cvScore) # change the cv parameter

    end = time.time()

    # print("Stochastic Gradient Descent Classifier Accuracy: {0:.2%}".format(accuracy_score(prediction, y_test)))
    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the sgdClf function with different value for cvScore(2, 3, 4, 5, 6)
for i in range(0,5):
    sgdClf(cvScore+i)
```

Резултат – Stochastic Gradient Descent Classifier		
cross validation: 2 Confusion Matrix: [[65 10] [2 37]] Accuracy: 89.47% Precision: 94.87% Recall: 78.72% F1 score: 86.05% Cross validation score: 78.73% (+/- 0.28%) Execution time: 0.030006 seconds	**cross validation: 3** **Confusion Matrix:** **[[65 10]** **[2 37]]** **Accuracy: 89.47%** **Precision: 94.87%** **Recall: 78.72%** **F1 score: 86.05%** **Cross validation score: 82.94% (+/- 7.84%)** **Execution time: 0.035994 seconds**	cross validation: 4 Confusion Matrix: [[73 2] [12 27]] Accuracy: 87.72% Precision: 69.23% Recall: 93.10% F1 score: 79.41% Cross validation score: 76.10% (+/- 14.18%) Execution time: 0.034997 seconds
cross validation: 5 Confusion Matrix: [[75 0] [17 22]] Accuracy: 85.09% Precision: 56.41% Recall: 100.00% F1 score: 72.13% Cross validation score: 84.90% (+/- 6.34%) Execution time: 0.038 seconds	cross validation: 6 Confusion Matrix: [[66 9] [6 33]] Accuracy: 86.84% Precision: 84.62% Recall: 78.57% F1 score: 81.48% Cross validation score: 80.19% (+/- 20.68%) Execution time: 0.082001 seconds	

Од добиените резултати можеме да заклучиме дека класификаторот со вредност за крос валидација 3 има најдобри резултати.

Accuracy: 89.47%
Precision: 94.87%
Recall: 78.72%
F1 score: 86.05%

2. **K Nearest Neighbors Classifier** – Овој експеримент е спроведен за различен број на соседи (5, 10, 15, 20, 25 и 30).

```
# K Nearest Neighbors Classifier
from sklearn.neighbors import KNeighborsClassifier
neighbors=5

def knn(neighbors):

    print("K Nearest Neighbors Classifier with n_neighbors:", neighbors)

    start = time.time()

    classifier = KNeighborsClassifier(n_neighbors = neighbors, metric = 'minkowski')
    classifierName = "K Nearest Neighbors Classifier"
    classifier.fit(X_train, y_train)
    prediction = classifier.predict(X_test)
    scores     = cross_val_score(classifier, X, y, cv=5)

    end = time.time()

    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the knn function with different number of neighbors(5, 10, 15, 20, 25, 30)
for i in range(0, 30, 5):
    knn(neighbors+i)
```

Резултат – K Nearest Neighbors Classifier		
n neighbors: 5 Confusion Matrix: [[69 6] [7 32]] Accuracy: 88.60% Precision: 82.05% Recall: 84.21% F1 score: 83.12% Cross validation score: 88.60% (+/- 6.96%) Execution time: 0.12 seconds	n neighbors: 10 Confusion Matrix: [[71 4] [8 31]] Accuracy: 89.47% Precision: 79.49% Recall: 88.57% F1 score: 83.78% Cross validation score: 89.13% (+/- 8.34%) Execution time: 0.083998 seconds	n neighbors: 15 Confusion Matrix: [[72 3] [8 31]] Accuracy: 90.35% Precision: 79.49% Recall: 91.18% F1 score: 84.93% Cross validation score: 88.61% (+/- 8.29%) Execution time: 0.068999 seconds

n_neighbors: 20 **Confusion Matrix:** **[[72 3]** **[7 32]]** **Accuracy: 91.23%** **Precision: 82.05%** **Recall: 91.43%** **F1 score: 86.49%** **Cross validation score: 89.31% (+/- 8.82%)** **Execution time: 0.076999 seconds**	**n_neighbors: 25** **Confusion Matrix:** **[[72 3]** **[7 32]]** **Accuracy: 91.23%** **Precision: 82.05%** **Recall: 91.43%** **F1 score: 86.49%** **Cross validation score: 89.31% (+/- 7.88%)** **Execution time: 0.075991 seconds**	n_neighbors: 30 Confusion Matrix: [[72 3] [8 31]] Accuracy: 90.35% Precision: 79.49% Recall: 91.18% F1 score: 84.93% Cross validation score: 89.31% (+/- 8.82%) Execution time: 0.071004 seconds

Од табелата погоре, можеме да заклучиме дека за **n_neighbors=20 и 25** добиваме најдобри вредности за Accuracy, Precision, Recall и F1 Score.

Accuracy: 91.23%
Precision: 82.05%
Recall: 91.43%
F1 score: 86.49%

3. **Random Forest Classifier** – Спроведени се 4 експерименти за овај класификатор со различни вредности за крос-валидацијата, како **criterion** е земено **'entropy'** и вредноста за естиматори е поставена на 50 (**n_estimators = 50**) што означува дека "шумата" е составена од 50 "дрва".

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier

cvScore=2

def random_forest(cvScore):
    print("Random Forest Classifier with cross validation:", cvScore)
    start = time.time()

    classifier = RandomForestClassifier(n_estimators = 50, criterion = 'entropy') # 50 trees
    classifierName = "Random Forest Classifier"
    classifier.fit(X_train, y_train)
    prediction = classifier.predict(X_test)
    scores = cross_val_score(classifier, X, y, cv=cvScore)

    end = time.time()

    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the random_forest function with different value for cvScore(2, 3, 4, 5)
for i in range(0,4):
    random_forest(cvScore+i)
```

Резултат – Random Forest Classifier	
cross validation: 2 Confusion Matrix: [[69 6] [5 34]] Accuracy: 90.35% Precision: 87.18% Recall: 85.00% F1 score: 86.08% Cross validation score: 87.70% (+/- 0.66%) Execution time: 0.424 seconds	**cross validation: 3** **Confusion Matrix:** **[[67 8]** **[3 36]]** **Accuracy: 90.35%** **Precision: 92.31%** **Recall: 81.82%** **F1 score: 86.75%** **Cross validation score: 89.28% (+/- 7.19%)** **Execution time: 0.551 seconds**
cross validation: 4 Confusion Matrix: [[69 6] [6 33]] Accuracy: 89.47% Precision: 84.62% Recall: 84.62% F1 score: 84.62% Cross validation score: 88.76% (+/- 8.42%) Execution time: 0.77101 seconds	cross validation: 5 Confusion Matrix: [[68 7] [4 35]] Accuracy: 90.35% Precision: 89.74% Recall: 83.33% F1 score: 86.42% Cross validation score: 89.84% (+/- 7.81%) Execution time: 0.86401 seconds

Од добиените резултати од експериментите донесуваме заклучок дека со крос-валидација=3 добиваме најдобри вредности за Accuracy, Precision, Recall и F1 Score .

Accuracy: 90.35%
Precision: 92.31%
Recall: 81.82%
F1 score: 86.75%

4. **Decision Tree Classifier** – Спроведени се 6 експерименти за вредност 2, 3, 4, 5, 6 и 7 за крос-валидацијата.

```
# Decision Tree Classifier

cvScore=2

def decision_tree(cvScore):
    print("Decision Tree Classifier with cross validation:", cvScore)
    start = time.time()

    classifier = DecisionTreeClassifier(criterion = 'entropy', splitter = 'best')
    classifierName = "Decision Tree Classifier"
    classifier.fit(X_train, y_train)
    prediction = classifier.predict(X_test)
    scores = cross_val_score(classifier, X, y, cv=cvScore)

    end = time.time()

    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the decision_tree function with different value for cvScore(2, 3, 4, 5, 6, 7)
for i in range(0,6):
    decision_tree(cvScore+i)
```

Резултат – Decision Tree Classifier		
cross validation: 2 Confusion Matrix: [[67 8] [4 35]] Accuracy: 89.47% Precision: 89.74% Recall: 81.40% F1 score: 85.37% Cross validation score: 85.06% (+/- 1.70%) Execution time: 0.024 seconds	cross validation: 3 Confusion Matrix: [[66 9] [4 35]] Accuracy: 88.60% Precision: 89.74% Recall: 79.55% F1 score: 84.34% Cross validation score: 87.52% (+/- 3.94%) Execution time: 0.03901 seconds	cross validation: 4 Confusion Matrix: [[67 8] [4 35]] Accuracy: 89.47% Precision: 89.74% Recall: 81.40% F1 score: 85.37% Cross validation score: 88.23% (+/- 5.34%) Execution time: 0.036006 seconds
cross validation: 5 **Confusion Matrix:** **[[67 8]** **[3 36]]** **Accuracy: 90.35%** **Precision: 92.31%** **Recall: 81.82%** **F1 score: 86.75%** **Cross validation score: 89.12% (+/- 6.52%)** **Execution time: 0.039001 seconds**	cross validation: 6 Confusion Matrix: [[67 8] [4 35]] Accuracy: 89.47% Precision: 89.74% Recall: 81.40% F1 score: 85.37% Cross validation score: 87.20% (+/- 7.71%) Execution time: 0.054995 seconds	cross validation: 7 Confusion Matrix: [[66 9] [4 35]] Accuracy: 88.60% Precision: 89.74% Recall: 79.55% F1 score: 84.34% Cross validation score: 87.02% (+/- 11.33%) Execution time: 0.058003 seconds

За крос-валидација 5 добиваме најдобри резултати од овој класификатор:

Accuracy: 90.35%
Precision: 92.31%
Recall: 81.82%
F1 score: 86.75%

5. **SVC - Support Vector Machines Classifier** – вкупно спроведени експерименти се 6, што значи дека од овој класификатор имаме различни резултати за различни мета параметри како што се **kernel, C, gamma** и **cross validation score.**

```
#  Support Vector Machines Classifier
from sklearn.svm import SVC

cvScore=2

def svc(cvScore, kernel, c, gamma):

# SVC
    print("Support Vector Machines Classifier with cross validation:", cvScore)
    start = time.time()

    classifier = SVC(kernel = kernel, C=c, gamma = gamma)
    classifierName = "SVC - Support Vector Machines Classifier"
    classifier.fit(X_train, y_train)
    prediction = classifier.predict(X_test)
    scores = cross_val_score(classifier, X, y, cv=cvScore)

    end = time.time()

    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the svc function
svc(4, 'rbf', 1.0, 'auto')
svc(3, 'rbf', 1.0, 'scale')
svc(4, 'rbf', 2.0, 'scale')

svc(2, 'linear', 1.0, 'auto')
svc(3, 'linear', 2.5, 'scale')
svc(4, 'linear', 0.8, 'scale')
```

Резултат – Support Vector Machines Classifier		
kernel = 'rbf', *C = 1.0,* *gamma = 'auto'* *cross validation: 4* Confusion Matrix: [[72 3] [38 1]] Accuracy: 64.04% Precision: 2.56% Recall: 25.00% F1 score: 4.65% Cross validation score: 64.50% (+/- 2.71%) Execution time: 0.1299 seconds	*kernel = 'rbf',* *C = 1.0,* *gamma = 'scale'* *cross validation: 3* Confusion Matrix: [[72 3] [8 31]] Accuracy: 90.35% Precision: 79.49% Recall: 91.18% F1 score: 84.93% Cross validation score: 88.23% (+/- 8.62%) Execution time: 0.03099 seconds	*kernel = 'rbf',* *C = 2.0,* *gamma = 'scale'* *cross validation: 4* Confusion Matrix: [[68 7] [3 36]] Accuracy: 91.23% Precision: 92.31% Recall: 83.72% F1 score: 87.80% Cross validation score: 88.23% (+/- 3.12%) Execution time: 1.833 seconds

kernel = 'linear', *C = 1.0,* *gamma = 'auto'* *cross validation: 2* Confusion Matrix: [[68 7] [3 36]] Accuracy: 91.23% Precision: 92.31% Recall: 83.72% F1 score: 87.80% Cross validation score: 88.23% (+/- 3.12%) Execution time: 1.748 seconds	***kernel = 'linear',*** ***C = 2.5,*** ***gamma = 'scale'*** ***cross validation: 3*** **Confusion Matrix:** **[[69 6]** **[3 36]]** **Accuracy: 92.11%** **Precision: 92.31%** **Recall: 85.71%** **F1 score: 88.89%** **Cross validation score: 90.51%** **(+/- 5.64%)** **Execution time: 7.2511 seconds**	*kernel = 'linear',* *C = 0.8,* *gamma = 'scale'* *cross validation: 4* Confusion Matrix: [[68 7] [3 36]] Accuracy: 91.23% Precision: 92.31% Recall: 83.72% F1 score: 87.80% Cross validation score: 90.52% (+/- 5.43%) Execution time: 3.8311 seconds

За kernel = 'linear', C = 2.5, gamma = 'scale' и cross validation: 3 Support Vector Machines Classifier да ва најдобар резултат:

Accuracy: 92.11%
Precision: 92.31%
Recall: 85.71%
F1 score: 88.89%

6. **NuSVC - Support Vector Machines Classifier** – Класификатор сличен на обичниот SVC-Support Vector Classifier, разликата е во тоа што овај класификатор има параметар што го контролира бројот на **support вектори** што се користат во експериментот.

```
#  Support Vector Machines Classifier
from sklearn.svm import NuSVC

cvScore=2

def nuSVC(cvScore, nu, kernel, gamma):

# NuSVC - Nu-Support Vector Classification. Similar to SVC but uses a parameter to control the number of support vectors.
    print("Support Vector Machines Classifier with cross validation:", cvScore)
    start = time.time()

    classifier = NuSVC(kernel = kernel, nu = nu, gamma = gamma)
    classifierName = "NuSVC - Support Vector Machines Classifier"
    classifier.fit(X_train, y_train)
    prediction = classifier.predict(X_test)
    scores = cross_val_score(classifier, X, y, cv=cvScore)

    end = time.time()

    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the nuSVC function
nuSVC(cvScore = 3, nu = 0.3, kernel = 'rbf', gamma = 'scale')
nuSVC(cvScore = 5, nu = 0.5, kernel = 'rbf', gamma = 'auto_deprecated')
nuSVC(cvScore = 4, nu = 0.7, kernel = 'linear', gamma = 'auto')
```

Резултат – Nu-Support Vector Machines Classifier		
kernel = 'rbf', ***nu = 0.3,*** ***gamma = 'scale'*** ***cross validation: 3*** **Confusion Matrix:** **[[72 3]** **[8 31]]** **Accuracy: 90.35%** **Precision: 79.49%** **Recall: 91.18%** **F1 score: 84.93%** **Cross validation score: 87 .35% (+/- 8.44%)** **Execution time: 0.055002 s econds**	*kernel = 'rbf',* *nu = 0.5,* *gamma = 'auto_deprecated'* *cross validation: 5* Confusion Matrix: [[69 6] [38 1]] Accuracy: 61.40% Precision: 2.56% Recall: 14.29% F1 score: 4.35% Cross validation score: 64 .84% (+/- 3.64%) Execution time: 0.219 seco nds	*kernel = 'linear',* *nu = 0.7,* *gamma = 'auto'* *cross validation: 4* Confusion Matrix: [[75 0] [20 19]] Accuracy: 82.46% Precision: 48.72% Recall: 100.00% F1 score: 65.52% Cross validation score: 79 .80% (+/- 9.67%) Execution time: 0.047002 s econds

Од изведените три експерименти од овој класификатор, забележуваме дека најдобриот резултат се добива со параметрите: **kernel = 'rbf' , nu = 0.3**, **gamma = 'scale'**, **cross validation: 3** со следните вредности:

Accuracy: 90.35%
Precision: 79.49%
Recall: 91.18%
F1 score: 84.93%

7. **LinearSVC - Support Vector Machines Classifier –** Линеарен support vector кој го спроведуваме со различни вредности за параметрите **C, loss('hinge' кое е стандардното SVM loss и 'squared_hinge' кое е квадратот на hinge loss), max_iter.**

```
#  Support Vector Machines Classifier
from sklearn.svm import LinearSVC

cvScore=2

def linearSVC(cvScore, C, loss, max_iter):

# LinearSVC
    print("Support Vector Machines Classifier with cross validation:", cvScore)
    start = time.time()

    classifier = LinearSVC(C = C, loss = loss, max_iter = max_iter)
    classifierName = "LinearSVC - Support Vector Machines Classifier"
    classifier.fit(X_train, y_train)
    prediction = classifier.predict(X_test)
    scores = cross_val_score(classifier, X, y, cv=cvScore)

    end = time.time()

    report(classifier, classifierName)
    Y_pred = classifier.predict(X_test)

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

```
# Calling the linearSVC function
linearSVC(cvScore = 2, C = 1.0, loss = 'squared_hinge', max_iter=900)
linearSVC(cvScore = 3, C = 0.5, loss = 'squared_hinge', max_iter=1000)
linearSVC(cvScore = 4, C = 1.0, loss = 'hinge', max_iter=1200)
```

Резултат – Linear-Support Vector Machines Classifier		
loss = 'squared_hinge', ***C = 1.0,*** ***max_iter = 900*** ***cross validation: 2*** **Confusion Matrix:** **[[73 2]** **[10 29]]** **Accuracy: 89.47%** **Precision: 74.36%** **Recall: 93.55%** **F1 score: 82.86%** **Cross validation score: 77 .51% (+/- 2.03%)** **Execution time: 0.082011 s econds**	*loss = 'squared_hinge',* *C = 0.5,* *max_iter = 1000* *cross validation: 3* Confusion Matrix: [[50 25] [0 39]] Accuracy: 78.07% Precision: 100.00% Recall: 60.94% F1 score: 75.73% Cross validation score: 86 .47% (+/- 2.79%) Execution time: 0.13502 se conds	*loss = 'hinge',* *C = 1.0,* *max_iter = 1200* *cross validation: 4* Confusion Matrix: [[75 0] [26 13]] Accuracy: 77.19% Precision: 33.33% Recall: 100.00% F1 score: 50.00% Cross validation score: 80 .84% (+/- 7.07%) Execution time: 0.1478 sec onds

За параметрите **loss = 'squared_hinge', C = 1.0, max_iter = 900** и **cross validation = 2** се добив а најдобриот резултат од Linear Support Vector Classifier со следните вредности:

Accuracy: 89.47%
Precision: 74.36%
Recall: 93.55%
F1 score: 82.86%

Споредба на најдобрите резултати од секој класификатор

Споредба на резултати						
Stochastic Gradient Descent Classifier	K Nearest Neighbors Classifier	Random Forest Classifier	Decision Tree Classifier	Support Vector Machines Classifier	Nu-Support Vector Machines Classifier	Linear-Support Vector Machines Classifier
Accuracy: 89.47% Precision: 94.87% Recall: 78.72% F1 score: 86.05%	Accuracy: 91.23% Precision: 82.05% Recall: 91.43% F1 score: 86.49%	Accuracy: 90.35% Precision: 92.31% Recall: 81.82% F1 score: 86.75%	Accuracy: 90.35% Precision: 92.31% Recall: 81.82% F1 score: 86.75%	Accuracy:92.11% Precision: 92.31% Recall: 85.71% F1 score: 88.89%	Accuracy: 90.35% Precision: 79.49% Recall: 91.18% F1 score: 84.93%	Accuracy: 89.47% Precision: 74.36% Recall: 93.55% F1 score: 82.86%

Според табелата погоре, лесно доаѓаме до заклучок дека **Support Vector Machines Classifier** го дава најдобриот резултат за овој дата сет со вредност:
Accuracy:92.11%, Precision: 92.31%, Recall: 85.71%, F1 score: 88.89% .

Овој класификатор е спроведен со следните параметри:
kernel = 'linear', C = 2.5, gamma = 'scale' и cross validation: 3

Подобрување на класификатори со GridSearchCV()

Тука ќе искористиме **GridSearchCV()** за да пробаме да ги тунираме параметрите на некој Класификатор за да дојдеме до најдобро решение со најдобри параметри.

1. Ќе се обидеме да направиме подобрување на **Random Forest Classifier**.

Соодветно како параметри во GridSearchCv() ги ставаме прво класификаторот, па потоа бр ојот на **n** естиматори и критериумот, scoring и n_jobs.

```
# Improving Random Forest Classifier

from sklearn.model_selection import GridSearchCV

X = dataSet.loc[:, features_mean]
y = dataSet.loc[:, 'diagnosis']
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

start = time.time()

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

classifier = GridSearchCV(RandomForestClassifier(),
                          parameters,
                          scoring = 'average_precision',
                          n_jobs=-1)

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

end = time.time()

print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(np.mean(scores), np.std(scores)*2))
report(classifier, "Random Forest Classifier")
print("Execution time: %s seconds \n" % "{0:.5}".format(end-start))

print("Best parameters: {0} \n".format(classifier.best_params_))
```

Како резултат на ова подобрување се добива следното:

Cross validation score: 94.65% (+/- 6.82%)
Classification report for the : Random Forest Classifier
Confusion Matrix:
[[70 5]
 [5 34]]
Accuracy: 91.23%
Precision: 87.18%
Recall: 87.18%
F1 score: 87.18%
Execution time: 200.84 seconds

Best parameters: {'criterion': 'gini', 'n_estimators': 78}

2. Ќе се обидеме да направиме подобрување на **Decision Tree Classifier**.

Соодветно како параметри во GridSearchCv() ги ставаме прво класификаторот, па потоа критериумот, splitter, scoring и n_jobs како последен параметар.

```
# Improving Decision Tree Classifier

start = time.time()

parameters = {'criterion':['gini', 'entropy'], 'splitter':['best', 'random']}

classifier = GridSearchCV(DecisionTreeClassifier(),
                          parameters,
                          scoring = 'average_precision',
                          n_jobs=-1)

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

end = time.time()

print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(np.mean(scores), np.std(scores)*2))
report(classifier, "Dedicion Tree Classifier")
print("Execution time: %s seconds \n" % "{0:.5}".format(end-start))

print("Best parameters: {0} \n".format(classifier.best_params_))
```

Резлутатот на подобрувањето:

Cross validation score: 77.64% (+/- 16.53%)
Classification report for the : Dedicion Tree Classifier
Confusion Matrix:
[[59 16]
 [5 34]]
Accuracy: 81.58%
Precision: 87.18%
Recall: 68.00%
F1 score: 76.40%
Execution time: 7.0676 seconds

Best parameters: {'criterion': 'entropy', 'splitter': 'random'}

Заклучок

Со вака изградениот модел на класификација можеме да видиме дека алгоритмот за Random Forest Classification дава најдобри резултати за нашата база на податоци. Што значи дека за да ние го избереме нашиот модел, секогаш треба да ја анализираме нашата база на податоци и потоа да го примениме нашиот модел за машинско учење и да ги споредуваме резултатите на секој класификатор. Со примена на ваков вид на програма може да се утврди раната дијагностика на ракот на дојка со што се подобрува шансата за преживување која е најбитна во случајот.

Референци

[1] Data set: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data
[2] Pandas: https://pandas.pydata.org/pandas-docs/stable/
[3] NumPy and Scipy: https://docs.scipy.org/doc/
[4] Seaborn: https://seaborn.pydata.org/
[5] Scikit-learn: https://scikit-learn.org/stable/modules/svm.html
[6] Wikipedia – Confusion Matrix: https://en.wikipedia.org/wiki/Confusion_matrix
[7] Developers.Google: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall