Praktisk Maskinlæring Oblig 2

En rapport av Stian Larsen

Datasett brukt i obligen baserer seg på pasienter som har / ikke har diabetes. Det er totalt 9 kolonner, med tilhørende 768 rader.

I min oppgave blir det å skape en modell som kan gjette på om en person har diabetes eller ei. Jeg har tatt i bruk 3 – 4 algortimer for å teste ut deres preg på oppgaven, og de aller fleste av modellene har like gode resultater, men noen få forskjeller ved seg.

Jeg hadde en god stund problemer med å skjønne hvorfor modellen min ikke helt leveret slik jeg hadde håpet på, til jeg tenkte på en ting... Jeg hadde ikke sjekket balansen på datasettet, jeg sjekket alt i forhold til at det ikke var noen «rare» datatyper det, jeg sjekket at ingen rader inneholdt «Not a Number», jeg sjekket også etter om noen rader hadde «null data» i form av at de var tomme. Men jeg glemte å sjekke om datasettet var i ubalanse.

Det viste det seg at det var!

Jeg fant ut av dette ved å lage 2 lister, og kjørte en FOR-løkka på target kolonna, og addet hvert av de 0 eller 1 tallene i hver sin liste. Se funksjonen under.

```
y_target_column = dataset["Outcome"]
test = []
hehe = []
# Check balance
for item in y_target_column:
    if item == 0:
        test.append(item)

    else:
        hehe.append(item)

print("0: ", len(test),
      "1 ere: ", len(hehe))
#   Output
#   0 (null): 500
#   1 (én): 268
```

Som man kan se på bildet fikk jeg en output som ga meg litt bakoversveis, mildt sagt.
Jeg fant fort ut at enten måtte jeg halvere antall rader som hadde «Outcome» = 0, eller øke antallet rander med «Outcome» = 1.

Dette går under begrepene «oversampling / undersampling», og i mitt tilfelle prøvde jeg ut både oversampling og undersampling med interessante resultater som jeg nå skal diskutere.

Oversampling

Til å teste ut virkningen av å balansere ut et ubalansert datasett ved bruk av oversampling, tok jeg i bruk et bibliotek kalt *imblearn*. Her har jeg tilgang til en modul som heter «RandomOverSampler», som i bunn og grunn balanserer ut datasettet ved å ta minioriteten, og høyne antallet rader for å matche det overdominerende.

Utfallet av å benytte dette viste seg å være svært liten. Jeg fikk så og si ingen forskjell i performance på accuracy, ei heller fikk jeg ikke noen bedre resultater på visualisering av confusion matrixen. Så det lønner seg ikke for min del å benytte dette videre.

Undersampling

Jeg benyttet samme bibliotek for å teste ut om undersampling ville skape et bedre utfall for modellen min. Funksjonen jeg tok i bruk heter RandomUnderSampling, og det tar inn x_train settet og y_train settet, og fjerner det dominerende antall rader for å matche minioriteten. Jeg fant ut her at modellen faktisk funket dårligere ved å ta i bruk denne metoden, i følge min confusion matrix fikk jeg flere falske påstander enn noen gang tidligere. Det samme kunne jeg også se på accuracy scoren til modellen, den viste meg en betraktelig dårligere accuracy score. Basert på disse to tilfellene har jeg valgt å fjerne både RandomOverSampling & RandomUnderSampling videre i oppgaven.

Modeller

For å skape modellene lagde jeg en liste som kalles for «Modeller». Slik jeg løsta oppgaven var å initialisere hver modell, og appende disse til listen «Modeller». Deretter kunne jeg kjøre en FOR-løkke på denne listen og kjøre programmet mot hver modell. Se bilde nedenfor for illustrasjon:

```
modeller = []

# ML models
# support Vector Machines
svm_model = svm.SVC(kernel="linear", C=1)
modeller.append(svm_model)

# check if props are missing numbers
missing_props = dataset.isna().mean(axis=0)
print(missing_props) # output = oK verified, no missing values.

# Decision trees
tree_model = tree.DecisionTreeClassifier(criterion="entropy", splitter="best")
modeller.append(tree_model)

# Naive Bayes Gaussian
naive_model = GaussianNB()
modeller.append(naive_model)

# XGBC calssifier
XGBC_model = XGBClassifier(eval_metric='mlogloss')
modeller.append(XGBC_model)
```

(Modeller initialisert: SVC, DTC, GNB, XGBC)

Support Vector Classifier

```
svm_model = svm.SVC(kernel="linear", random_state=42)
```

SVC er en modell som baserer seg på regressjon implementert ved bruk av libsvm. Denne er ment for bruk på classification datasett, og den brukes ofte ved ansiktsgjenkjenning, tekst-gjenkjenning o.l

Min modell har flere parametere man kan tune, og jeg har testen en del av de.

Kernel

Kernel er typen algoritme man velger å bruke ved å instansiere denne klassifisereren. Jeg har gått for «Linear» av den grunn at den har bedre performance enn de andre kernelene jeg har prøvd ut. Blant noen å nevne har jeg testet ut «sigmoid», «rbf» og «poly». Resultatene er henoldsvis:

1. sigmoid: En accuracy på 0.54
2. rbf: En accuracy på 0.79
3. poly: En accuracy på 0.81
4. linear: En accuracy variert fra 0.81 til 0.84416

Decision Tree Classifier

Decision Tree Classifier er en algoritme innenfor «supervised learning» i maskinlæring. Denne algoritmen er en av de mest populære ved siden av SVC. Den benytter et sett av regler den følger for å avgjøre endelige resultater.

Criterion

Criterion er en funksjon som måler kvaliteten på splittingen for datasettet. Det finnes 3 type funksjoner man kan teste ut på DTC, og det er «entropy», «log_loss» og «Gini». I de tilfelle jeg brukte Gini og log_less fikk jeg dårligere resultater enn ved å bruke entropy, så jeg valgte å bruke entropy som hovedfunksjon for splitting.

1. entropy: En accuracy på 0.6948051948051948
2. log_loss: En accuracy på 0.68182
3. gini: En accuracy på 0.67532

Som du kan se er det kun snakk om maks ca. 2% accuracy som skiller den beste fra den verste, men jeg holdt meg videre som sagt til entropy.

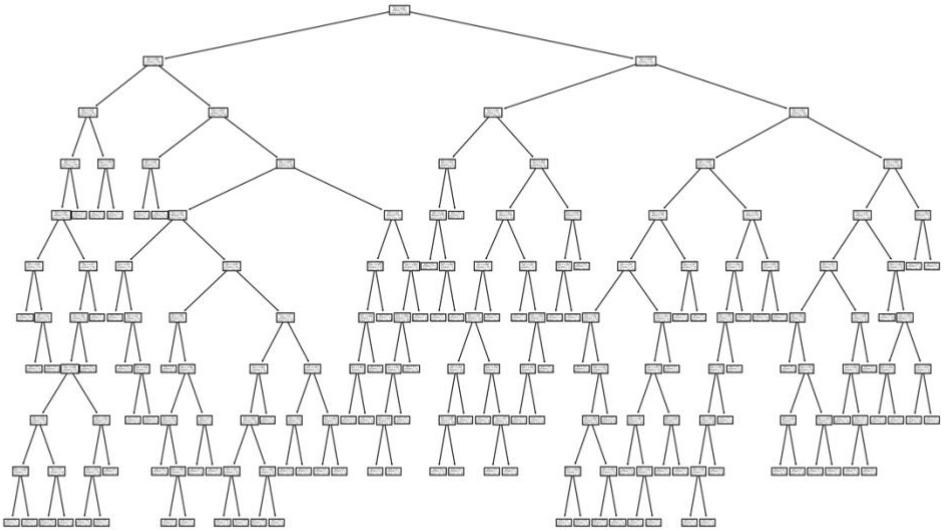
Max_depth

Det fantes også flere parametere man kunne tune, jeg la til max_depth for å maksimere antall noder den kunne lage i treet sitt, og jeg testet først ut med max_depth = 20, og fikk et

overraskende godt resultat basert på dette. Jeg forsket så videre på dette, og resultatene er som følger:

(criterion = «entropy», max_depth = 20, splitter = «best») ➔ Accuracy på: 0.73377
(criterion = «entropy», max_depth = 5, splitter = «best») ➔ Accuracy på: 0.78571
(criterion = «entropy», max_depth = 10, splitter = «best») ➔ Accuracy på: 0.68831

Resultatet av den beste modellen her ble et tre som ser slik ut:



Gaussian Naïve Bayes

Naïve Bayes er en statistisk fremgangsmetode for å kalkulere seg frem til sannsynligheten for et utfall. I mitt prosjekt på oblig 2 har jeg tatt i bruk Gaussan Naive bayes for å teste ut om dette ville være et bedre alternativ enn de andre overnevnte.

Teoremet går ut på å sjekke sannsynligheten for utfallet basert på en gitt situasjon. Formelen som ofte brukes ser slik ut:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

Den kan beskrives slikl: A & B er hendelser, og funksjonen sier enkelt og greit «Hva er sannsynligheten for A, gitt hendelse B.» Den kan da regne ut sannsynligheten for at utfallet A vil skje, basert på at hendelse B finner sted.

Bayes teoremet kan kan da brukes til å lage en klassifikerende modell:

$$P(y|X) = \frac{P(X|y).P(X)}{P(y)}$$

Det finnes 3 forskjellige typer Naive Bayes klassifiseringer, og de er:

1. Gaussian
2. Bernoulli
3. Multinomial

Jeg har tatt i bruk Gaussan.

Uten noen form for tweaking av parameterne og tune modellen, gir den et ganske greit resultat enn hva vi har sett tidligere:
Accuracy på test-settet: 0.81169

Ved å endre på parameteren var_smoothing kunne vi endre på porsjonen av variansen mellom alle inputene for bedre stabilitet i beregningene. Jeg endret på den defaulte verdien på: ***1e-9***.

Jeg endret denne til 1e-5, og fikk en accuracy på 0.78, jeg testet så med 1e-3 og fikk acc: 0.69. Så basert på dette lot jeg den stå på 1e-9. Jeg testet for så vidt med 1e-10 og 1e-20 bare for å se. Men den makset ut på en accuracy på 0.8111xxx, så det var uten hell.

XGB Classifier

Denne modellen har jeg sett frem til å prøve ut, da jeg har hørt mye bra om den. Den kan tunes veldig mye skal jeg tro på andre.

XGB står for EXTREME GRADIENT BOOSTING, og er egentlig et decision tree, bare i form av et parrallelt tree som bruker gradient boosting.

Jeg testet ut modellen på datasettet og sammenlignet resultatene, og de er interessante:

Uten tuning ser resultatet på datasettet slik ut:

Accuracy: 0.73377

Jeg testet kun ut den defaulte modellen da jeg egt ikke skulle ha mer enn 3 modeller. Men var på grunn av nysgjerrigheten at jeg testet ut modell nr 4, men fikk overraskende dårlig resultat med denne modellen.

Jeg tunet den og testet ut attributten; Eval_metric = «mlogloss» uten noen endring. Gav meg på modellene her.

Resultater

Basert på forskjellige classification modeller har jeg funnet ut noen resultater om hver av de.

Slik jeg gjennomførte testene mot slutten, var at jeg kjørte en FOR-løkke igjennom listen med alle modellene, og fikk full oversikt i konsollen av hver og en modell, side om side. På denne måten slapp jeg å repetere koden min, og jeg fikk den samme infoen på alle sammen.

Samtidig ble det også gjort 5-fold cross-validation for hver eneste modell som ble kjørt i loopen. Scoren kan man se i bildene nedenunder som representerer konsoll-oiutputen for alle modellene, splittet opp for synligere preg.

Man kan feks se i SVC modellen en cross-validation som viser:

```
Cross Validation Performance results:
[0.75609756 0.76422764 0.79674797 0.77235772 0.73770492]
Cross Validation mean: 0.7654271624683461 accuracy with a standard deviation of 0.01942178494933315
```

Ved test av cross-validation får vi en lavere score gjennomsnittling på train settet. Det bildet du nettopp så over, vil også være nedenfor for alle modellene:

```python
# Test the accuracy of the model by looping through the array of models we created earlier
for model in modeller:
    print(f"Model: {model}")
    # Fit and train the model with the x train and y train set.
    model.fit(x_train, y_train)
    # predict the results with x_test set
    y_prediction = model.predict(x_test)
    # get the accuracy score
    accuracy = accuracy_score(y_test, y_prediction)
    # print results
    print(f"Accuracy of {model} is {accuracy}\n")
    # check for true positives, false negatives and true negatives & false positives.
    confusion = confusion_matrix(y_test, y_prediction)
    crossValidation = cross_val_score(model, x_train, y_train, cv=5)
    print(f"Confusion Matrix of {model} is: \n{confusion}")
    print(f"Cross Validation Performance results: \n"
          f"{crossValidation}\n"
          f"Cross Validation mean: {crossValidation.mean()} accuracy with a standard "
          f"deviation of {crossValidation.std()}")
    print('Train data accuracy: ', round(model.score(x_train, y_train), 5))
    print('Test data accuracy: ', round(model.score(x_test, y_test), 5))
    print("\n\n\n\n")


"""
    For previewing Decision Tree nodes..
    plt.figure(figsize=(20, 20))
    plot_tree(model)
    plt.show()
"""
```

Resultatene er som følger:

Fra konsollen:

SVM:

```
Model: Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(kernel='linear', random_state=44))])
Accuracy of Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(kernel='linear', random_state=44))]) is 0.8441558441558441

Confusion Matrix of Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(kernel='linear', random_state=44))]) is:
[[97 10]
 [14 33]]
Cross Validation Performance results:
[0.75609756 0.76422764 0.79674797 0.77235772 0.73770492]
Cross Validation mean: 0.7654271624683461 accuracy with a standard deviation of 0.01942178494933315
Train data accuracy:  0.75896
Test data accuracy:  0.84416
```

(SVM)

DTC:

```
Model: DecisionTreeClassifier(criterion='entropy', max_depth=10)
Accuracy of DecisionTreeClassifier(criterion='entropy', max_depth=10) is 0.6753246753246753

Confusion Matrix of DecisionTreeClassifier(criterion='entropy', max_depth=10) is:
[[79 28]
 [22 25]]
Cross Validation Performance results:
[0.72357724 0.69918699 0.71544715 0.69918699 0.71311475]
Cross Validation mean: 0.7101026256164201 accuracy with a standard deviation of 0.009565606574650613
Train data accuracy:  0.96417
Test data accuracy:  0.67532
```

(DTC)

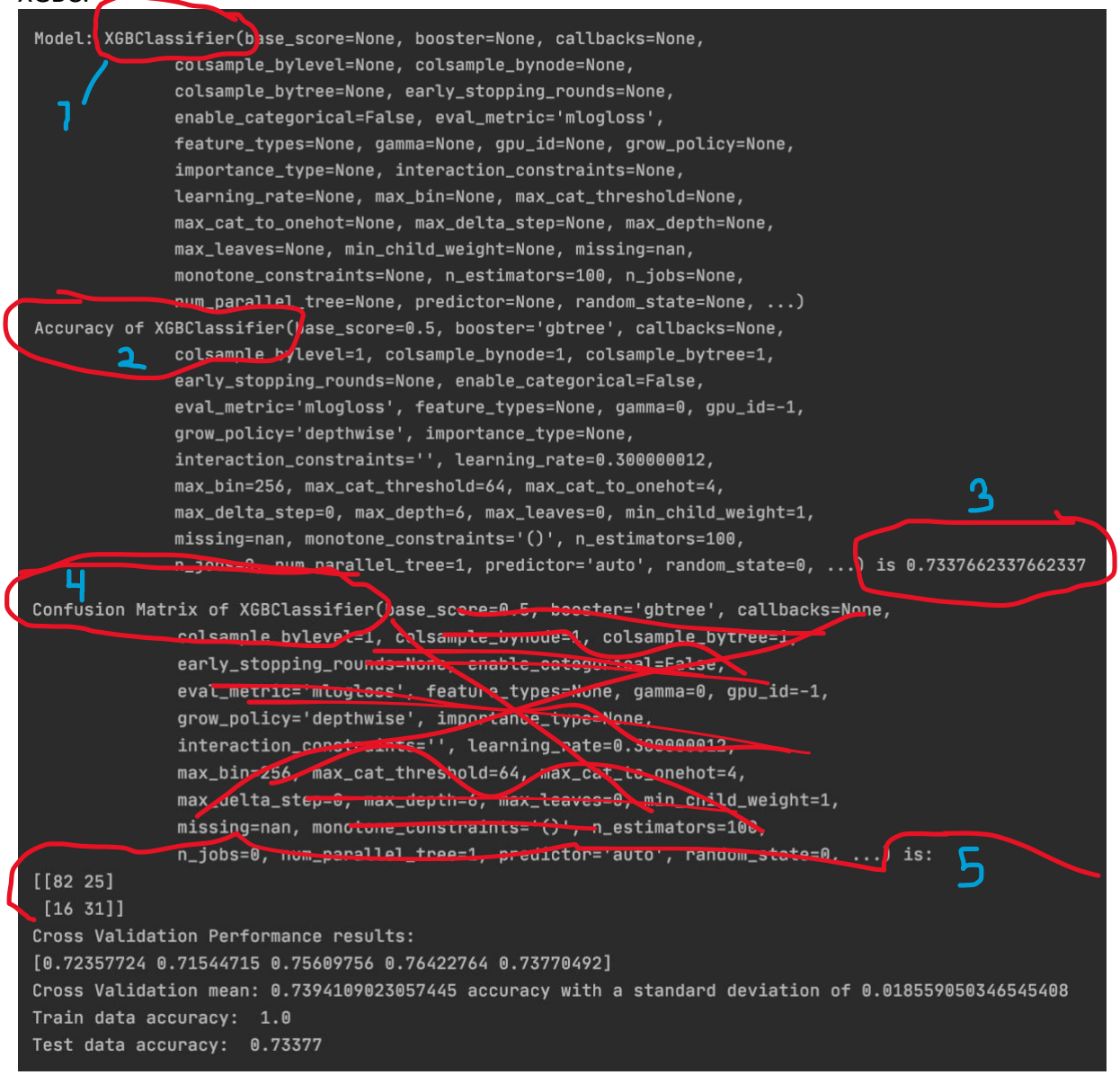
Gaussan NB:

```
Model: GaussianNB()
Accuracy of GaussianNB() is 0.8116883116883117

Confusion Matrix of GaussianNB() is:
[[92 15]
 [14 33]]
Cross Validation Performance results:
[0.74796748 0.70731707 0.78861789 0.72357724 0.68852459]
Cross Validation mean: 0.7312008529921364 accuracy with a standard deviation of 0.03472075274913848
Train data accuracy:  0.73616
Test data accuracy:  0.81169
```

(Gaussan NB)

XGBC:



(XGBC)

Oppsummering:
Blant modellene er jeg overbevist om at SVC modellen er den beste for settet mitt. De alle leverer nogenlunde resultater, de kunne selvfølgelig vært bedre hadde jeg gått for kanskje et annet datasett, men det var spennende å jobbe med modellene.