Praktisk Maskinlæring

Oblig 3 skrevet av Stian Larsen

Regression

I denne delen er jeg blitt bedt om å velge ut et datasett som skal brukes til å teste ut regression-metoden for å predikere datasettet.

Datasettet jeg skal jobbe med er:

Concrete_data

Og det består av features:

- Cement
- Blast Furnace Slag
- Fly Ash
- Water
- Superplasticizer
- Coarse Aggregate
- Fine Aggregate
- Age

Og target:

- Strength

Antall rader:

- 1001 inkl. Header

Data prosessering

Litt teori før jeg starter demonstrasjon.

Til å starte med går jeg igjennom datasettet for å bli litt kjent med det. Se hvordan det ser ut, om noe skiller seg ut fra det vanlige. Dette er nødvendig for å sørge for at modellen vi skal bruke på datasettet senere, ikke blir overrasket av noe som ikke burde være der, dette kan for eksempel være at dataene ikke har riktig format som forventet, at det er skyhøye tall og store variasjoner i maksimum og minimums-dataene. Da kan vi normalisere dataene om dette er tilfelle, og vi kan luke ut de radene, eventuelt kolonnene som ikke gjør annet enn å støye til for modellen vi skal bruke.

Vi vil med andre ord sørge for at modellen får et godt utgangspunkt, og da er første steg som nevnt over, veldig viktig.

Videre kan vi sjekke balansen i datasettet, hvordan er dataene spredt i forhold til hverandre? Er det oversampling eller undersampling? Dette må vi luke ut, eventuelt gjøre noe med.

Fra tidligere har jeg erfart at undersampling har ført til bedre resultater enn hva oversampling har gjort, så dette må sjekkes opp i.

Utforsking av datasettet:

Jeg startet med å se på dataene ved å kjøre funksjonene:

- .head():

```
First 5 rows of the normalized dataset:
   Cement  Blast Furnace Slag  Fly Ash  ...  Fine Aggregate  Age  Strength
0   540.0                 0.0      0.0  ...           676.0   28     79.99
1   540.0                 0.0      0.0  ...           676.0   28     61.89
2   332.5               142.5      0.0  ...           594.0  270     40.27
3   332.5               142.5      0.0  ...           594.0  365     41.05
4   198.6               132.4      0.0  ...           825.5  360     44.30
```

- .describe():

```
Some general info about the dataset:
            Cement  Blast Furnace Slag  ...          Age     Strength
count  1000.000000         1000.000000  ...  1000.000000  1000.000000
mean    282.141200           72.710000  ...    46.192000    35.696640
std     105.082271           86.065907  ...    64.036097    16.828158
min     102.000000            0.000000  ...     1.000000     2.330000
25%     194.700000            0.000000  ...     7.000000    23.520000
50%     272.800000           20.000000  ...    28.000000    33.945000
75%     356.750000          142.500000  ...    56.000000    46.230000
max     540.000000          359.400000  ...   365.000000    82.600000
```

- .info():

  Denne funksjonen returnerte så mye data at det ligna på samme info fra .head().

Steg etter utforskingen: :

i) Det første jeg gjorde var å normalisere dataene med min-max normalisering.

ii) Jeg hadde også så mange rader at jeg måtte legge til low_memory=»False» ved initialiseringen av datasettet! Fjernet senere rader så det ble akkurat 1000 rader..

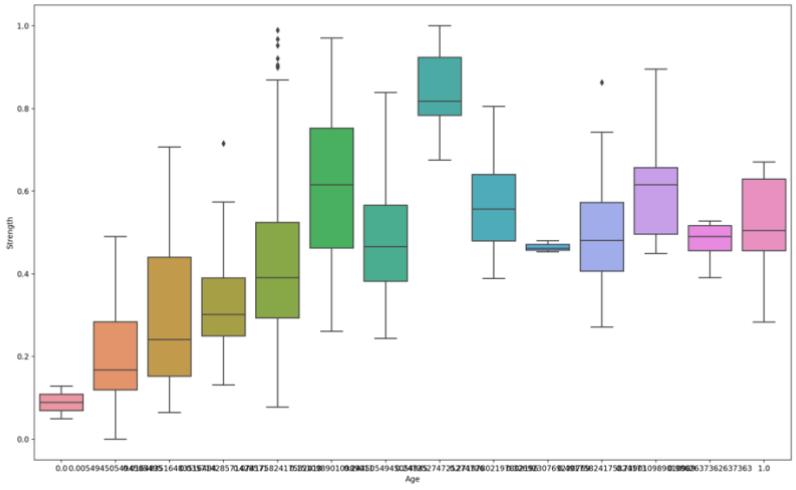
iii) Etter å ha sjekket balansen var jeg fornøyd med fordelingen, så lot denne være.

iv) Splitting av dataene. Jeg gikk for 75/25 splitting. Sjekket dette med en enkel funksjon og fikk denne «shapen»:

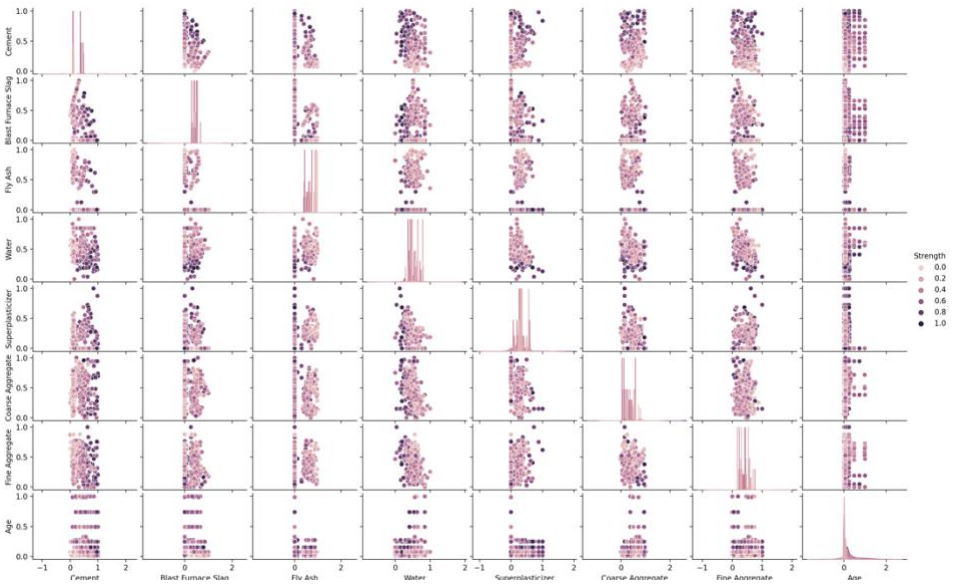
```
Shape of datasets:
x_train: (750, 8)
x_test: (250, 8)
y_train: (750,)
y_test: (250,)
```

(Denne viser et resultat av splittingen på 75/25. 75% er forbeholdt training-settet, imens 25% er til test settet. Vi kan se at train-settet har 750 rader, mot 250 hos test-settet.)

v) Jeg har kjørt boxplot og pairplot for å visualisere dataene litt mer, dette gjør det mulig å se dataene vi jobber med litt bedre:



(Boxplot, x = Age & Y = strength.)



(Pairplot)

Modeller

Som i oblig 2, lagde jeg en modeller [] array hvor jeg appender inn alle modellene jeg initialiserer. Dette gjør det enklere, og penere i konsollen. Slik er fremgangsmetoden:

```
"""         Initiate Models         """
modeller = []

"""   Linear Regression   """
# Default startup
linear_regression_model = LinearRegression()
modeller.append(linear_regression_model)
```

(Dette gjøres for hver modell, Gjelder også Tuning, hver tuning er sin egen modell og blir lagt til i listen.)

Som caption forklarer, så lager jeg alle modellene på denne måten, og mot slutten når jeg tuna parameterne så addet jeg disse modellene også.

Selve funksjonen som tester modellene er en FOR løkke som kjøres på hver modell i listen:

```
167  scores = {}
168  for model in modeller:
169      desired_prediction = 1.0
170      # Fit the model
171      model.fit(x_train, y_train)
172      # Generate score for training set
173      training_score = model.score(x_train, y_train)
174      # Generate score for Test set
175      test_score = model.score(x_test, y_test)
176      print(f"\n\n"
177            f"Model used: {model}\n"
178            f"Training score: {round(training_score, 3)}\n"
179            f"Test score: {round(test_score, 3)}\n"
180            f"Avvik mellom Train & Test set: {round((training_score - test_score), 3)}\n"
181            f"Vi er {round((desired_prediction - test_score), 3)} fra {desired_prediction}\n"
182            f"med et resultat på: {((round(test_score, 3) * 100))}%"
183      )
184      predicted_percent = (round(test_score, 3) * 100) # Predicted percentage..
185      scores.update({model: round(predicted_percent, 3)}) # Adding the model and the percent into a dict.
186
187  # Print out table of the models performance Sorted:
188  print("\n\n{:<55} {:<10}".format("Model", "Percent"))
189  scores = dict(sorted(scores.items(), key=lambda item: item[1], reverse=True))
190  for bestmodel in scores:
191      print("{:<55} {:<10}".format(str((bestmodel)), scores[bestmodel]))
192
193
```

Linear Regression

Lineær regresjon omfatter mye matematikk, men enkelt og greit viser den til forholdet mellom avhengige variabler og uavhengige variabler. Basert på dette forholdet kan vi bruke regresjon til å «spå» / «gjette» på hva et neste datapunkt for den avhengige variabelen kan være, basert på uavhengige verdier vi mater inn.

Resultat:

```
Model used: LinearRegression()
Training score: 0.622
Test score: 0.6
Avvik mellom Train & Test set: 0.022
Vi er 0.4 fra 1.0
med et resultat på: 60.0%
```

Tuning

Ved første øyekast fant jeg ut at Linear Regression har ganske få parametere å tune, men jeg testet ut å endre på «fit_intercept» til True/False, uten noen forbedring..

Fant deretter ut at man kan teste ut modulen: SGDRegressor, som er en lineær regresjonsmodell, også kalt: Stochastic Gradient Descent.

I forsøk på å finne beste parametere, måtte jeg først finne ut hvilke parametere jeg hadde å jobbe, slik:

Jeg kjørte i gang en instanse av SGDRegressor(), og tok getParams() funksjon på denne instansen. Da fikk jeg opp alle parametere tilgjengelig.

GridSearch

Resultat:
parameter generated: {'warm_start': False, 'verbose': 4, 'validation_fraction': 0.9, 'shuffle': False, 'penalty': 'elasticnet', 'n_iter_no_change': 4, 'max_iter': 850, 'loss': 'squared_error',

'learning_rate': 'adaptive', 'l1_ratio': 0.5, 'fit_intercept': True, 'eta0': 0.04, 'epsilon': 1.2, 'early_stopping': False, 'average': False, 'alpha': 0.0005}

```
# Tuning
alpha = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.005, 0.01, 0.05, 0.1]
max_depth = list((range(0, 150, 5)))
bools = [True, False]
epsilon = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5, 2]
eta0 = [0.01, 0.02, 0.03, 0.04, 0.05]
min_samples_split = list(range(2, 10))
l1_ratio = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
loss = ["squared_error", "huber", "epsilon_insensitive", "squared_epsilon_insensitive"]
learning_rate = ["constant", "optimal", "invscaling", "adaptive"]
max_iter = list(range(50, 1000, 50))
n_iter_no_change = [1, 2, 3, 4, 5]
penalty = ["l2", "l1", "elasticnet"]
validation_fraction = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
verbose = [0, 2, 4, 5]

gridSearch = {...}
Linear_tuning = RandomizedSearchCV(estimator=sgd_Regressor, param_distributions=gridSearch, error_score="raise")
Linear_tuning.fit(x_train, y_train)
```

Basert på koden over kan jeg benytte forslaget til parametere i modellen, og teste ut om jeg får en bedre score mot datasettet.

Før tuning:

```
Model used: SGDRegressor()
Training score: 0.344
Test score: 0.337
Avvik mellom Train & Test set: 0.007
Vi er 0.663 fra 1.0
med et resultat på: 33.7%
```

(Ganske dårlig score spør du meg, kanskje denne regresjonsmetoden ikke er helt ment for mitt sett? Jeg tester uansett videre.)

Etter tuning:

```
Model used: SGDRegressor(alpha=0.0008, early_stopping=True, epsilon=0.2, eta0=0.03,
             l1_ratio=0.8, learning_rate='constant', max_iter=900,
             n_iter_no_change=3, shuffle=False, validation_fraction=0.2,
             verbose=2)
Training score: 0.597
Test score: 0.618
Avvik mellom Train & Test set: -0.021
Vi er 0.382 fra 1.0
med et resultat på: 61.8%
```

(Veldig fornøyd med å ha doblet presisjonen til modellen! Score på: 0.618, opp fra 0.337 er veldig bra spør du meg.)

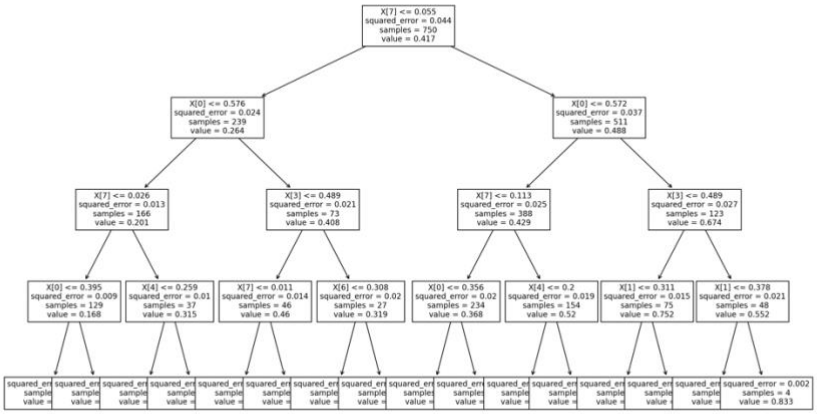
Decision Tree Regression

“

Decision tree Regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output

“
— Wikipedia

Jeg trenet opp modellen min med default values, og fikk dette treet opp:



Dette vil komme til nytte når vi skal måle scoren på modellen vår for de predikerte tallene. Da vil den benytte dette treet som hjelp for å guide og spå hvor et punkt skal befinne seg hen.

Resultat

```
Model used: DecisionTreeRegressor()
Training score: 0.995
Test score: 0.841
Avvik mellom Train & Test set: 0.154
Vi er 0.159 fra 1.0
med et resultat på: 84.1%
```

Tuning

For å tune Decision Tre´et så fulgte jeg forrige steg, satte opp et GridSearch og fant de beste parameterne for modellen, slik:

```python
# Tuning
splitter = ["best", "random"]
max_depth = list((range(0, 150, 5)))
bools = [True, False]
min_samples_split = list(range(2, 10))
min_samples_leaf = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.2, 1.5, 2]
min_weight_fraction_leaf = [0.1, 0.2, 0.3, 0.4, 0.5]
max_leaf_nodes = list(range(2, 20, 1))
criterion = ["poisson", "absolute_error", "friedman_mse", "squared_error"]
gridSearch = {
            "splitter": splitter,
            "max_depth": max_depth,
            "min_samples_split": min_samples_split,
            "min_samples_leaf": min_samples_leaf,
            "min_weight_fraction_leaf": min_weight_fraction_leaf,
            "max_leaf_nodes": max_leaf_nodes,
            "criterion": criterion
}
decision_tree_tuning = RandomizedSearchCV(estimator=decision_tree, param_distributions=gridSearch)
decision_tree_tuning.fit(x_train, y_train)
```

Som vi så i resultatet over bildet, så fikk vi en score på 84.1%, før tuning,

Etter tuning:

```
Model used: DecisionTreeRegressor(criterion='absolute_error', max_depth=65,
                      max_leaf_nodes=12, min_samples_split=5,
                      min_weight_fraction_leaf=0.1)
Training score: 0.575
Test score: 0.599
Avvik mellom Train & Test set: -0.024
Vi er 0.401 fra 1.0
med et resultat på: 59.9%
```

(Her fikk jeg overraskende nok et dårligere resultat. Prøvde å kjøre grid 2 ganger ekstra og oppdaterte parameterne uten noen hell her.)

Etter dette kjørte jeg også en test bare av å kun oppdatere parameteren «criterion», slik så resultatet ut:

```
Model used: DecisionTreeRegressor(criterion='absolute_error')
Training score: 0.993
Test score: 0.884
Avvik mellom Train & Test set: 0.109
Vi er 0.116 fra 1.0
med et resultat på: 88.4%
```

(criterion = absolute_error // score = 88.4%)

```
Model used: DecisionTreeRegressor()
Training score: 0.995
Test score: 0.874
Avvik mellom Train & Test set: 0.121
Vi er 0.126 fra 1.0
med et resultat på: 87.4%
```

(criterion = default) // score = 87.4%

```
Model used: DecisionTreeRegressor(criterion='poisson')
Training score: 0.995
Test score: 0.868
Avvik mellom Train & Test set: 0.126
Vi er 0.132 fra 1.0
med et resultat på: 86.8%
```

(criterion = poisson // score = 86.8%)

```
Model used: DecisionTreeRegressor(criterion='friedman_mse')
Training score: 0.995
Test score: 0.864
Avvik mellom Train & Test set: 0.13
Vi er 0.136 fra 1.0
med et resultat på: 86.4%
```

(criterion = friedman_mse) // score = 86.4%

Ridge Regression

Resultat

```
Model used: Ridge()
Training score: 0.611
Test score: 0.616
Avvik mellom Train & Test set: -0.006
Vi er 0.384 fra 1.0
med et resultat på: 61.6%
```

Radom Forest Regression

Denne har jeg hørt mye godt om fra klassekamerater, og tenkte jeg skulle teste den ut. Ble raskt overbevist om at denne er en skikkelig råtass:

Resultat

```
Model used: RandomForestRegressor()
Training score: 0.983
Test score: 0.903
Avvik mellom Train & Test set: 0.08
Vi er 0.097 fra 1.0
med et resultat på: 90.3%
```

Tuning

```python
# == The tuning ==
max_depth = list((range(0, 150, 5)))
bools = [True, False]
min_samples_split = list(range(2, 10))
criterion = ["squared_error", "absolute_error", "poisson"]
n_iter_no_change = [1, 2, 3, 4, 5]
verbose = [0, 2, 4, 5]
n_estimators = [5, 20, 50, 100]
min_samples_leaf = list(range(1, 50, 5))
min_weight_fraction_leaf = [0.1, 0.2, 0.3, 0.4, 0.5]
max_features = ["sqrt", "log2"]
max_leaf_nodes = list(range(2, 20, 1))
min_impurity_decrease = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
n_jobs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


gridSearchFR = {
            "n_estimators": n_estimators,
            "verbose": verbose,
            "warm_start": bools,
            "criterion": criterion,
            "min_samples_split": min_samples_split,
            "min_samples_leaf": min_samples_leaf,
            "min_weight_fraction_leaf": min_weight_fraction_leaf,
            "max_features": max_features,
            "max_leaf_nodes": max_leaf_nodes,
            "min_impurity_decrease": min_impurity_decrease,
            "bootstrap": bools,
            "n_jobs": n_jobs,
            "max_depth": max_depth
}
randomForest_tuner = RandomizedSearchCV(estimator=randomForest, param_distributions=gridSearchFR, error_score="raise")
randomForest_tuner.fit(x_train, y_train)
```

Anbefaling:

```python
# Creating the best model
randomForest_tuner_bestModel = RandomForestRegressor(
    n_estimators=randomForest_tuner.best_params_['n_estimators'],
    max_depth=randomForest_tuner.best_params_['max_depth'],
    warm_start=randomForest_tuner.best_params_['warm_start'],
    verbose=randomForest_tuner.best_params_['verbose'],
    n_jobs=randomForest_tuner.best_params_['n_jobs'],
    min_weight_fraction_leaf=randomForest_tuner.best_params_['min_weight_fraction_leaf'],
    min_samples_split=randomForest_tuner.best_params_['min_samples_split'],
    bootstrap=randomForest_tuner.best_params_['bootstrap'],
    min_samples_leaf=randomForest_tuner.best_params_['min_samples_leaf'],
    min_impurity_decrease=randomForest_tuner.best_params_['min_impurity_decrease'],
    max_leaf_nodes=randomForest_tuner.best_params_['max_leaf_nodes'],
    max_features=randomForest_tuner.best_params_['max_features'],
    criterion=randomForest_tuner.best_params_['criterion'],

)
modeller.append(randomForest_tuner_bestModel)
# Score from the random search random forest regressor
```

Etter tuning:

```
Model used: RandomForestRegressor(bootstrap=False, criterion='absolute_error', max_depth=5,
                      max_features='log2', max_leaf_nodes=10,
                      min_samples_split=5, min_weight_fraction_leaf=0.3,
                      n_estimators=20, n_jobs=4, verbose=4, warm_start=True)
Training score: 0.408
Test score: 0.367
Avvik mellom Train & Test set: 0.041
Vi er 0.633 fra 1.0
med et resultat på: 36.7%
```

(Overraskende dårlig etter tuning..)

Konklusjon

Etter å ha prøvd ut mange forskjellige varianter, har jeg laget denne oversikten:

```
Model                                                   Percent
RandomForestRegressor()                                 90.2
DecisionTreeRegressor(criterion='absolute_error')       79.7
DecisionTreeRegressor(criterion='friedman_mse')         78.2
DecisionTreeRegressor(criterion='poisson')              77.7
DecisionTreeRegressor()                                 77.4
LinearRegression()                                      58.3
Ridge()                                                 57.9
```

Så kommer de lange tunede modellene, litt rart formatert, beklager det:

```
SGDRegressor(alpha=0.0008, early_stopping=True, epsilon=0.2, eta0=0.03,
             l1_ratio=0.8, learning_rate='constant', max_iter=900,
             n_iter_no_change=3, shuffle=False, validation_fraction=0.2,
             verbose=2) 57.6
DecisionTreeRegressor(criterion='absolute_error', max_depth=65,
                      max_leaf_nodes=12, min_samples_split=5,
                      min_weight_fraction_leaf=0.1) 57.2
RandomForestRegressor(bootstrap=False, criterion='absolute_error', max_depth=5,
                      max_features='log2', max_leaf_nodes=10,
                      min_samples_split=5, min_weight_fraction_leaf=0.3,
                      n_estimators=20, n_jobs=4, verbose=4, warm_start=True) 36.7
```

1. SGDRegressor // Score: 57.6%
2. DecisionTreeRegressor // Score: 57.2%
3. RandomForestRegresson // Score: 36.7%