Jakub Chwastek
Mariusz Witkowski

Spis treści

1 Tytuł raportu - Electric Motor Temperature

2 Wstęp - motywacja, cele

Pod lupę bierzemy bazę danych dotyczącą temperatury silników elektrycznych w zależności od natężenia prądu itp. Celem badania jest sprawdzenie, w jakich okolicznościach silniki indukcyjne i bezszczotkowe mogą się przegrzewać, a w jakich będą działać bez zarzutu.

3 Opis danych - struktura zbiorów, opis zmiennych, pochodzenie

Baza zawiera takie dane jak: temperatura otoczenia stojana, temperatura płynu chłodzącego, napięcie **d-component**, napięcie **q-component**, prędkość obrotowa silnika, moment obrotowy, natężenie **d-component**, natężenie **q-component**, temperatura wirnika, jarzma stojana, zęba stojana i uzwojenia stojana. Warto zauważyć, że badanie zawiera 81 osobnych sesji pomiarowych - oznacza to, że będziemy musieli je analizować osobno. Wszystkie wartości zgodnie z opisem autora danych zostały znormalizowane względem pewnej wartości. Plik *csv* pobrany ze strony *kaagle.com* jest w pełni uporządkowany i gotowy jest do analizy bez jego uprzedniego formatowania. Fragment danych wywołany poprzez funkcję *.head()* wygląda następująco:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding	profile_id
0	-0.752143	-1.118446	0.327935	-1.297858	-1.222428	-0.250182	1.029572	-0.245860	-2.522071	-1.831422	-2.066143	-2.018033	4
1	-0.771263	-1.117021	0.329665	-1.297686	-1.222429	-0.249133	1.029509	-0.245832	-2.522418	-1.830969	-2.064859	-2.017631	4
2	-0.782892	-1.116681	0.332771	-1.301822	-1.222428	-0.249431	1.029448	-0.245818	-2.522673	-1.830400	-2.064073	-2.017343	4
3	-0.780935	-1.116764	0.333700	-1.301852	-1.222430	-0.248636	1.032845	-0.246955	-2.521639	-1.830333	-2.063137	-2.017632	4
4	-0.774043	-1.116775	0.335206	-1.303118	-1.222429	-0.248701	1.031807	-0.246610	-2.521900	-1.830498	-2.062795	-2.018145	4

Rys. 3.1: Skrawek danych

Jak widać na rysunku 3.1, wszystkie wartości zmiennych zostały już znormalizowane. Dane zostały również podzielone za pomocą zmiennej *profile id* na dane testowe (*profile id* $= 65$) oraz dane uczące - wszystkie pozostałe.

4 Opis procesu przygotowywania danych do analizy - kolejne kroki

Jak już zostało wspomniane w poprzednim punkcie, pobrany plik *csv* nie wymaga żadnej obróbki - analizę można rozpoczynać na gotowym pliku.

5 Analiza danych - przyjęte założenia, krótki opis metod i obranej metodologii analizy

Dane zawarte w pobranym przez nas pliku występują w formacie *float* oraz *int*. Dzięki czemu nie będziemy musieli w żaden sposób kategoryzować bądź ingerować w typ danych:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 998070 entries, 0 to 998069
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   ambient         998070 non-null  float64
 1   coolant         998070 non-null  float64
 2   u_d             998070 non-null  float64
 3   u_q             998070 non-null  float64
 4   motor_speed     998070 non-null  float64
 5   torque          998070 non-null  float64
 6   i_d             998070 non-null  float64
 7   i_q             998070 non-null  float64
 8   pm              998070 non-null  float64
 9   stator_yoke     998070 non-null  float64
 10  stator_tooth    998070 non-null  float64
 11  stator_winding  998070 non-null  float64
 12  profile_id      998070 non-null  int64
dtypes: float64(12), int64(1)
memory usage: 99.0 MB
```

Rys. 5.2: info()

Ich próbka, wywołana za pomocą funkcji *.head()* jest następująca:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding	profile_id
0	-0.752143	-1.118446	0.327935	-1.297858	-1.222428	-0.250182	1.029572	-0.245860	-2.522071	-1.831422	-2.066143	-2.018033	4
1	-0.771263	-1.117021	0.329665	-1.297686	-1.222429	-0.249133	1.029509	-0.245832	-2.522418	-1.830969	-2.064859	-2.017631	4
2	-0.782892	-1.116681	0.332771	-1.301822	-1.222428	-0.249431	1.029448	-0.245818	-2.522673	-1.830400	-2.064073	-2.017343	4
3	-0.780935	-1.116764	0.333700	-1.301852	-1.222430	-0.248636	1.032845	-0.246955	-2.521639	-1.830333	-2.063137	-2.017632	4
4	-0.774043	-1.116775	0.335206	-1.303118	-1.222429	-0.248701	1.031807	-0.246610	-2.521900	-1.830498	-2.062795	-2.018145	4

Rys. 5.3: head()

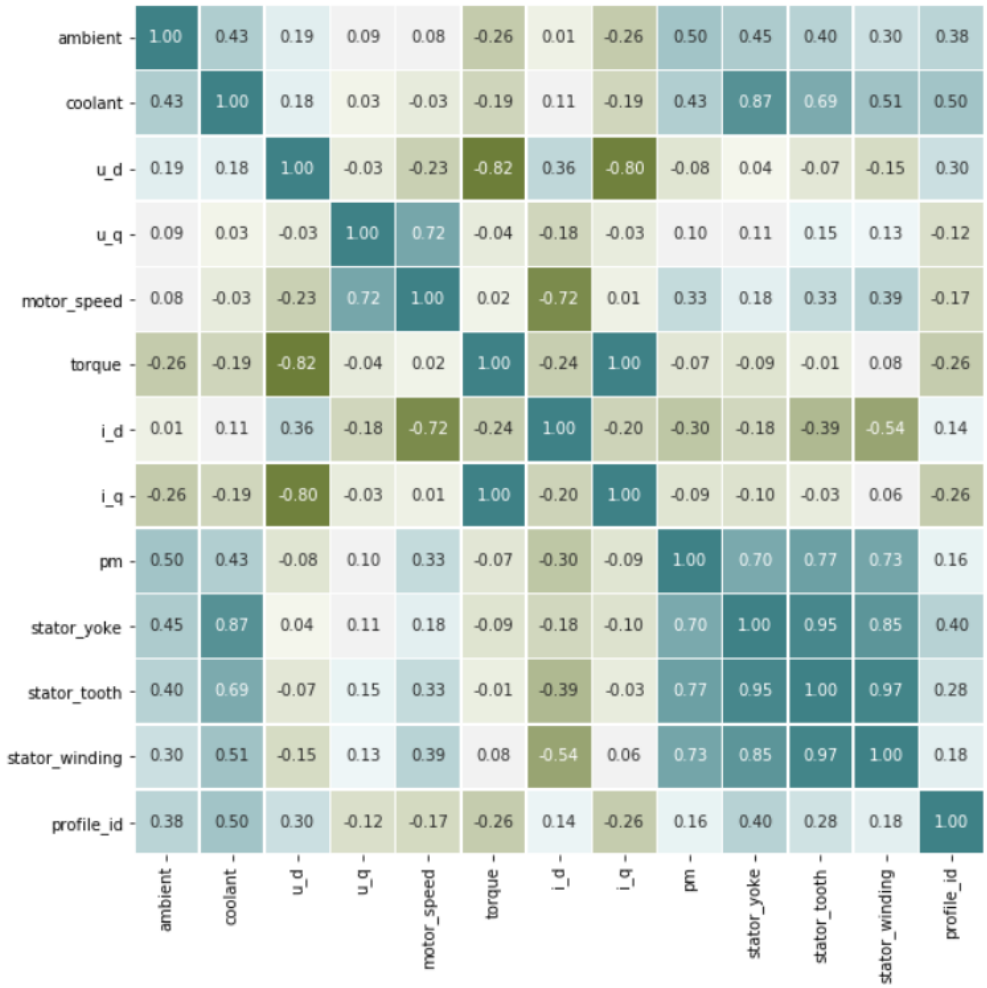
Tabela zawierająca średnie, najmniejsze i największe wartości jest następująca:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding	profile_id
count	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00	998070.00
mean	-0.00	0.00	0.00	-0.01	-0.01	-0.00	0.01	-0.00	-0.00	0.00	-0.00	-0.00	50.73
std	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	22.07
min	-8.57	-1.43	-1.66	-1.86	-1.37	-3.35	-3.25	-3.34	-2.63	-1.83	-2.07	-2.02	4.00
25%	-0.60	-1.04	-0.83	-0.93	-0.95	-0.27	-0.76	-0.26	-0.67	-0.75	-0.76	-0.73	32.00
50%	0.27	-0.18	0.27	-0.10	-0.14	-0.19	0.21	-0.19	0.09	-0.06	0.01	0.01	56.00
75%	0.69	0.65	0.36	0.85	0.85	0.55	1.01	0.50	0.68	0.70	0.77	0.73	68.00
max	2.97	2.65	2.27	1.79	2.02	3.02	1.06	2.91	2.92	2.45	2.33	2.65	81.00

Rys. 5.4: describe()

Jak widać w rzędzie *mean*, dane są już ustandaryzowane - średnia danych w każej kolumnie równa jest 0. W dalszej części raportu podzielimy ten dane ze względu na temperaturę wirnika - dokonamy kategoryzacji. Tablica korelacji natomiast wygląda jak poniżej:

Zmienna *pm* oznacza temperaturę wirnika - i wokół tej zmiennej będziemy tworzyli nasz model.



Rys. 5.5: corr()

Z tablicy korelacji widać bardzo wyraźnie zależność ze zmiennymi *stator yoke*, *stator tooth*, *stator winding* oraz *motor speed* czyli kolejno: temperatura jarzma, zębów oraz uzwojenia stojana i prędkosc silnika. Jest to jak najbardziej prawidłowy wynik, części tego samego urządzenia będą miały wysoce skorelowaną temperaturę. Prędkość silnika natomiast odwrotnie skorelowana jest z natężeniem płynącym przez **d-component** oraz proporcjonalnie z napięciem **q-component**.

6 Modelowanie danych - przyjęte założenia, krótki opis metod i obranej metodologii budowania modeli

Do zamodelowania danych wykorzystaliśmy dwie metody - regresję liniową oraz kategoryzację**??**, które dokładniej zostaną opisane poniżej:

6.1 Regresja liniowa

Na osi odciętych naszej funkcji umieścimy zmienną *pm*, a więc temperaturę wirnika. W tym celu połączymy wartości z kolumny *pm* z kolumną *profile id*. Jest to niezbędne do dalszej analizy, aby

oddzielić dane testowe (*profile id* = 65) od danych treningowych. W takim przypadku otrzymujemy:

	pm	profile_id
0	-2.522071	4
1	-2.522418	4
2	-2.522673	4
3	-2.521639	4
4	-2.521900	4

Rys. 6.6: head()

W wektorze osi rzędnych natomiast znajdować będą się wszystkie zmienne poza badaną *pm*. Osiągnięto to za pomocą funkcji:

$$x = temp.iloc[:, temp.columns! = "pm"]$$

Następnie, za pomocą funkcji

$$fit(x, y]$$

ze zbioru *sklearn.linear_model.LinearRegression* dopasowujemy model linowy do naszych zmiennych x oraz y. Atrybut *Intercept_* pozwala na ukazanie Independent **term in the linear model**:

$$array([1.20950222 * 10^{-1}, -3.05533376 * 10^{-13}])$$

Następnie, za pomocą atrybutu *coef_* ukazujemy estymaty współczynników funkcji liniowej modelu:

```
array([[ 2.28917957e-01, -2.43903379e-01, -2.32919886e-02,
        -3.46511415e-01,  3.28004478e-01,  5.40084836e-02,
         1.76829902e-01, -3.45091956e-02, -1.54700914e+00,
         4.52467007e+00, -2.26632173e+00, -2.40611203e-03],
       [-5.75211861e-13,  1.39780585e-13, -4.10004231e-14,
         2.71798162e-14,  4.37874592e-15,  8.30643999e-15,
         1.14292519e-14, -1.08768627e-14,  7.01357632e-14,
        -9.79654319e-14,  3.91876177e-14,  1.00000000e+00]])
```

Rys. 6.7: coef()

W następnej kolejności wyznaczamy średnie wartości rzędnych, tj. wszystkich danych poza kolumną *"pm"* za pomocą komendy *x_new = x.mean().values.reshape(1,-1)* otrzymując przy tym tablicę:

```
array([[-3.90549208e-03,  4.72251019e-03,  4.78041829e-03,
        -5.68972252e-03, -6.33550799e-03, -3.33285043e-03,
         6.04297096e-03, -3.19401557e-03,  6.09140300e-04,
        -2.20773772e-03, -3.93476002e-03,  5.07320008e+01]])
```

Rys. 6.8: xnew

Wartość przewidywana: Jak widać na Rys. 6.9 wartość przewidywana y dla średnich wartości x,

Rys. 6.9: ynew

wyniosła -4.39579466e-03, co jest dobrym znakiem, ponieważ wartość średnia zmiennej pm dla danych nie ustandaryzowanych wynosiła -0.004396.

Teraz należy obliczyć ustandaryzowana wartość zmiennych w wektorze rzędnych:

$$x\_std = \frac{x - x.mean(axis = 0)}{x.std(axis = 0)}$$

Jak widać na poniższym rysunku, standaryzacja przeszła zgodnie z założeniami (na rysunku nie zawarto kilku zmiennych w celu pomieszczenia czytelnego wycinka danych):

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	stator_yoke
count	9.980700e+05	9.980700e+05	9.980700e+05	9.980700e+05	9.980700e+05	9.980700e+05	9.980700e+05	9.980700e+05	9.980700e+05
mean	-3.610023e-13	2.148898e-13	-6.967515e-15	5.266386e-16	-2.328349e-14	3.632011e-13	-2.871728e-15	1.645897e-14	-1.226009e-13
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-8.629360e+00	-1.430605e+00	-1.663684e+00	-1.851459e+00	-1.363518e+00	-3.349632e+00	-3.255190e+00	-3.345430e+00	-1.833373e+00
25%	-5.996010e-01	-1.040127e+00	-8.329064e-01	-9.195573e-01	-9.443952e-01	-2.641374e-01	-7.631066e-01	-2.546065e-01	-7.470907e-01
50%	2.719318e-01	-1.814696e-01	2.633205e-01	-9.390934e-02	-1.337458e-01	-1.842994e-01	2.081014e-01	-1.872726e-01	-5.777421e-02
75%	6.953601e-01	6.444247e-01	3.544627e-01	8.563197e-01	8.588634e-01	5.516582e-01	1.008947e+00	5.035053e-01	6.960048e-01
max	2.991584e+00	2.637917e+00	2.274781e+00	1.795005e+00	2.028006e+00	3.026640e+00	1.055956e+00	2.923482e+00	2.445983e+00

Rys. 6.10: x_std

Kolejnym krokiem jest wyznaczenie wartości *y_pred* za pomocą funkcji *predict()* ze zbioru *sklearn.linear_model.LinearRegression()*:

	pm	profile_id
0	-2.522071	4
1	-2.522418	4
2	-2.522673	4
3	-2.521639	4
4	-2.521900	4
5	-2.522203	4
6	-2.522538	4
7	-2.522844	4
8	-2.522808	4
9	-2.522677	4

Rys. 6.11: y_pred

Następnie możemy wyznaczyć rezultaty uzysk

6.2 Kategoryzacja - pandas.cut()

Po stworzeniu modelu regresji liniowej postanowiliśmy skategoryzować dane pod względem wartości 'pm', przy pomocy funkcji z biblioteki pandas o nazwie cut(). Stworzyliśmy trzy kategorie: low, normal i high. Wielkości tych kategorii widoczne są na rys.

```
normal    670024
low       171324
high      156722
Name: overheat, dtype: int64
```

Rys. 6.12: kategorie

Następnie stworzyliśmy model, który na podstawie danych wejściowych, określał w jakiej kategorii będzie znajdował się pm. Do tego zadania wybraliśmy algorytm K najbliższych sąsiadów. Algorytm ten przypisuje daną do najbliższej jej kategorii. Po wyuczeniu modelu, wykonane zostały testy sprawdzające wiarygodność modelu. Wyniki widoczne są na rys.

```
In [290]: pd.Series(fit_classifier(knn, X_ucz, X_test, yk_ucz, yk_test))

ACC_ucz     0.998215
ACC_test    0.997069
P_ucz       0.998215
P_test      0.997069
R_ucz       0.998215
R_test      0.997069
F1_ucz      0.998215
F1_test     0.997069
dtype: float64
```

Rys. 6.13: wyniki

Wszystkie testy pokazują zaskakująco dobre rezultaty, co wzbudza pewne podejrzenia. Ostatecznie większym zaufaniem obdarzyłbym model regresji liniowej.

7 Rezultaty, wnioski i ich dyskusja

W celu sprawdzenia otrzymanych wyników regresji liniowej, wykonane zostały trzy różne testy. Pierwszym z nich był r score, a dokładniej r^2, czyli współczynnik determinacji. Współczynnik ten przyjmuje wartości od zera do jeden, gdzie jedynka oznacza, że wszystkie przewidziane przez nas wyniki, odpowiadały w pełni wynikom rzeczywistym. Kolejnym testem było sprawdzenie błędu średniokwadratowego, czyli MSE. Ostatnim testem było sprawdzenie średniego błędu bezwzględnego, czyli MAE. MSE oraz MAE chcemy aby były jak najmniejsze. Poniżej na rysunku 7.10 widać wyniki tych testów, dla zbioru podzielonego funkcją "train test split" w proporcjach 80 do 20.

	r_score_u	r_score_t	MSE_u	MSE_t	MAE_u	MAE_t
Reg. liniowa	0.886356	0.42367	0.113966	0.070112	0.18198	0.152878

Rys. 7.14: oceny

Wynik r2 dla zbioru testowego pozostawia wiele do życzenia, ale pozostałe dwa testy pokazują zadawalające wyniki. Następnie w celu dodatkowej weryfikacji uzyskanych rezultatów, przeprowadziłem walidację krzyżową dzieląc zbiór na 4 części i sprawdzając wyniki dla każdego możliwego zestawienia zbiorów uczących i zbioru testowego. Wyniki tej walidacji widoczne są na Rys.7.12

Dzięki takiej weryfikacji, możemy stwierdzić, że poprzedni niezadawalający wynik r2 dla zbioru

In [51]:
```
lista_wynik
```

	r_score_u	r_score_t	MSE_u	MSE_t	MAE_u	MAE_t
0	0.727196	0.732732	0.191293	0.400544	0.341521	0.482906
1	0.777489	0.697412	0.241444	0.150787	0.374861	0.307588
2	0.768477	0.760496	0.234852	0.211617	0.367127	0.364103
3	0.805749	0.568077	0.214880	0.269781	0.346823	0.409522

In [52]:
```
lista_wynik.describe()
```

	r_score_u	r_score_t	MSE_u	MSE_t	MAE_u	MAE_t
count	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000
mean	0.769728	0.689679	0.220617	0.258182	0.357583	0.391030
std	0.032497	0.085079	0.022577	0.106620	0.015952	0.074096
min	0.727196	0.568077	0.191293	0.150787	0.341521	0.307588
25%	0.758157	0.665078	0.208983	0.196410	0.345497	0.349975
50%	0.772983	0.715072	0.224866	0.240699	0.356975	0.386813
75%	0.784554	0.739673	0.236500	0.302472	0.369060	0.427868
max	0.805749	0.760496	0.241444	0.400544	0.374861	0.482906

Rys. 7.15: cross validation

testowego, dzielonego metodą train test split, mógł być spowodowany dobraniem wyjątkowo trudnych danych testowych. Z drugiej strony błędy MSE oraz MAE okazały się być wyższe niż pokazał to poprzedni test, nie mniej jednak wciąż są dopuszczalne.

Zastanawiając się nad tym co może wpływać na wyniki, przeprowadziłem estymacje używając metody OLS, czyli Ordinary Least Squares. Jej wyniki można zaobserwować na Rys. 7.13. Obliczony przez nią wynik r2 jest zbliżony do średniego wyniku z walidacji krzyżowej. Jednak możemy z tej analizy wynieść znacznie więcej informacji, takich jak zależność wyniku od poszczególnych zmiennych. Mówi nam o tym kolumna oznaczona wyrażeniem "$P > |t|$", a interpretować należy ją następująco: Jeśli wartość danej zmiennej w tej kolumnie jest mniejsza niż 0.05, to oznacza, że ta zmienna ma duży wpływ na wynik. Im większa wartość, tym mniejszy wpływ ma dana zmienna na końcowy rezultat. Tutaj widać, że zmienna torque mogłaby zostać pominięta i nie wpłynęło by to na końcowy rezultat.

```
pm ~ ambient+coolant+u_d+u_q+motor_speed+torque+i_d+i_q+stator_yoke+stator_tooth+stator_winding
                            OLS Regression Results
==============================================================================
Dep. Variable:                     pm   R-squared:                       0.772
Model:                            OLS   Adj. R-squared:                  0.772
Method:                 Least Squares   F-statistic:                 3.075e+05
Date:                Fri, 12 Jun 2020   Prob (F-statistic):               0.00
Time:                        14:26:58   Log-Likelihood:            -6.7373e+05
No. Observations:              998070   AIC:                          1.347e+06
Df Residuals:                  998058   BIC:                          1.348e+06
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept         -0.0010      0.000     -2.027      0.043      -0.002   -3.19e-05
ambient            0.2174      0.001    380.938      0.000       0.216       0.219
coolant           -0.2661      0.003    -87.324      0.000      -0.272      -0.260
u_d               -0.0364      0.001    -34.057      0.000      -0.039      -0.034
u_q               -0.3437      0.001   -323.057      0.000      -0.346      -0.342
motor_speed        0.3330      0.002    191.352      0.000       0.330       0.336
torque            -0.0020      0.008     -0.254      0.800      -0.017       0.013
i_d                0.1795      0.001    126.906      0.000       0.177       0.182
i_q                0.0182      0.007      2.508      0.012       0.004       0.032
stator_yoke       -1.5581      0.009   -164.914      0.000      -1.577      -1.540
stator_tooth       4.5505      0.012    382.386      0.000       4.527       4.574
stator_winding    -2.2783      0.006   -386.486      0.000      -2.290      -2.267
==============================================================================
Omnibus:                    45738.278   Durbin-Watson:                   0.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           106946.532
Skew:                           0.284   Prob(JB):                         0.00
Kurtosis:                       4.500   Cond. No.                         66.1
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Rys. 7.16: OLS