Predicting Road Accident Risk

Algorithms number 1, gruppe 69, 02.11.2025

# **1: BESKRIV PROBLEMET**

## **OMFANG / *SCOPE***

Oppgavens hovedproblem og datagrunnlag er gitt via Kaggles konkurranse i “Playground Series”, sesong 5, episode 10 “Predicting Road Accident Risk”. [[1]](https://www.zotero.org/google-docs/?LWOoSt) Formålet med prosjektet er å utvikle en maskinlæringsmodell som kan forutsi sannsynligheten for trafikkulykker på ulike typer veier. Modellen skal ta inn et sett av forklaringsvariabler og beregne en predikert risiko mellom 0 og 1.

Målet med oppgaven vil være å undersøke hvordan maskinlæring kan bidra til å peke ut strekninger og situasjoner med økt risiko. Selv om det er gitt med syntetiske data, er dette datapunkter som en kan tenke seg er lett tilgjengelig, som værforhold, vei-type, hastighet etc.. Og vi antar at dersom vi kan bruke maskinlæringsmodeller til å predikere basert på dette datasettet, kunne vi i en større oppgave sett på å predikere med real-world data.

Utenfor oppgavens scope, kunne dette brukes til korrekt fordeling av ressurser under trafikkplanlegging, vedlikehold av veinettet, forsikringsselskapers risikovurdering, politikontroller etc. …

## **METRIKKER**

Vi vil måle dataene med RMSE/RMSD (root mean square deviation/error) [[2]](https://www.zotero.org/google-docs/?ZmfC0Y)

Vi vil sammenligne vår endelige løsningen med 2 ulike sammenligningsgrunnlag

### Enkel RandomForestRegressor

Vi bruker RandomForest med så enkel data manipulasjon som mulig.

* Vi sjekker at alle boolean verdier er representert som int
* Alle kategoriske verdier er representert med int
* Vi bruker standardscaler på tallverdiene [[3]](https://www.zotero.org/google-docs/?7RLgIa)
* Vi bruker standardparametere for RandomForestClassifier[[4]](https://www.zotero.org/google-docs/?WeiI0x)

Dette gir oss en RMSE på **0.05935**

### Sammenligner med konkurransen leder

Konkurransen leder vil forandres over tid. Det vil ikke være et mål å nødvendigvis slå lederen av en konkurranse med tusenvis av deltakere. Men vi tenker dette vil gi oss en god pekepinn på hva som er mulig.

Per 26.10 var lederens resultat **0.05536**

### Mål for suksessfullt resultat

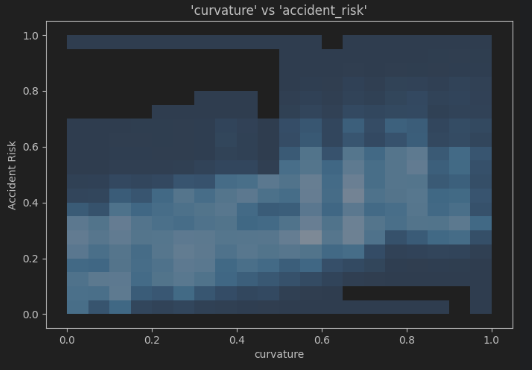
Vi håper å bygge en modell som scorer mellom **0.05935 og 0.05536** (eller bedre), der lavere score er bedre.

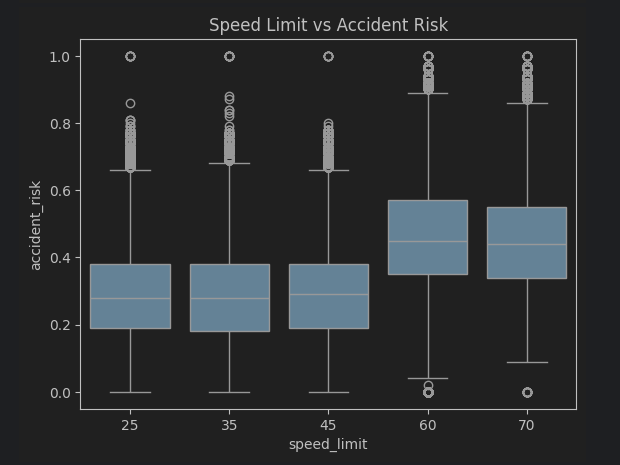
# **2: DATA**

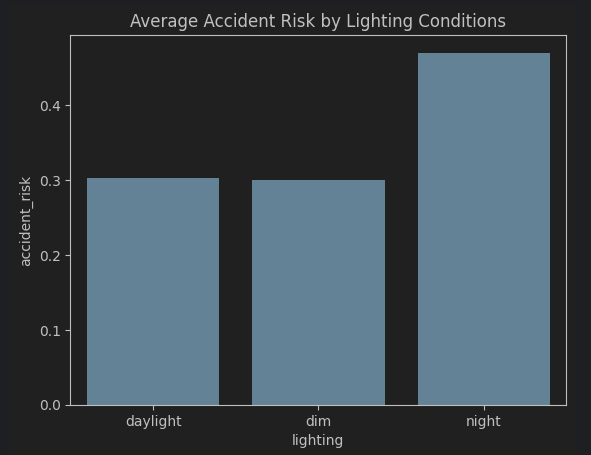
Datasettet er syntetiske data skapt for konkurransen, og for å være et nyttig datasett for å lære viktige konsepter innen maskinlæring. Datasettet kommer som en csv (komma separert fil). Den inneholder 14 kolonner som kan påvirke ulykkesrisiko i ulik grad.

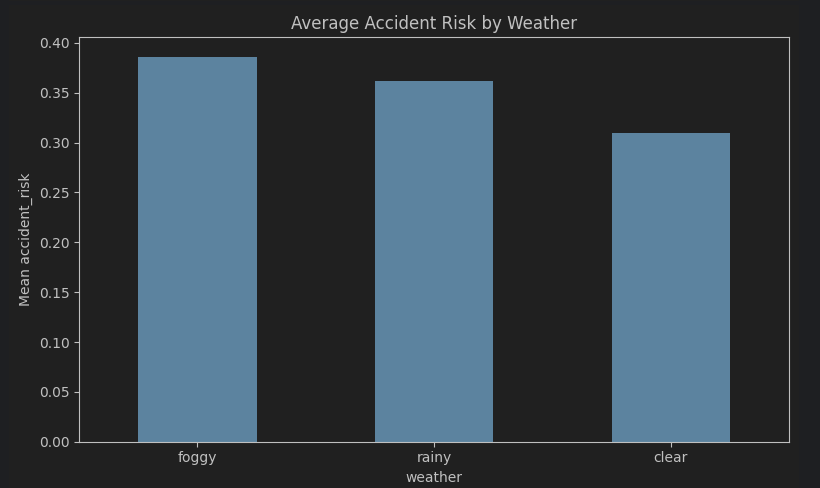
Under undersøkelsen av datasettet har vi satt opp en rekke grafer for å undersøke korrelasjoner i datasettet, og undersøke ideer for videre feature engineering.

## Utvalg interessante grafer









### Feature Engineering

#### Preparer features

* Jeg skiller ut boolean, kategoriske, og numeriske features
* Booleans sikrer jeg er tolket som boolean (ikke string “False”/”True”)
* Kategoriske features går gjennom sckitlearns OneHotEncoder [[5]](https://www.zotero.org/google-docs/?f3PW93)
* Numeriske features går gjennom StandardScaler[[3]](https://www.zotero.org/google-docs/?tS8Sos)
  + z = (x - u) / s

#### Finne engineered features

I første omgang satt jeg opp en baseline LGBM modell for å jobbe videre med feature engineering. Da fikk jeg en baseline modell som scorer RMSE: **0.0556454**, denne vil jeg jobbe videre med.

Angående features så jeg gjennom analysen av data at jeg vil fokusere på å finne features som er relatert til nattkjøring med andre risikofaktorer, tidspunkt relatert til andre til andre risikofaktorer, samt en rekke andre simple interaksjoner.

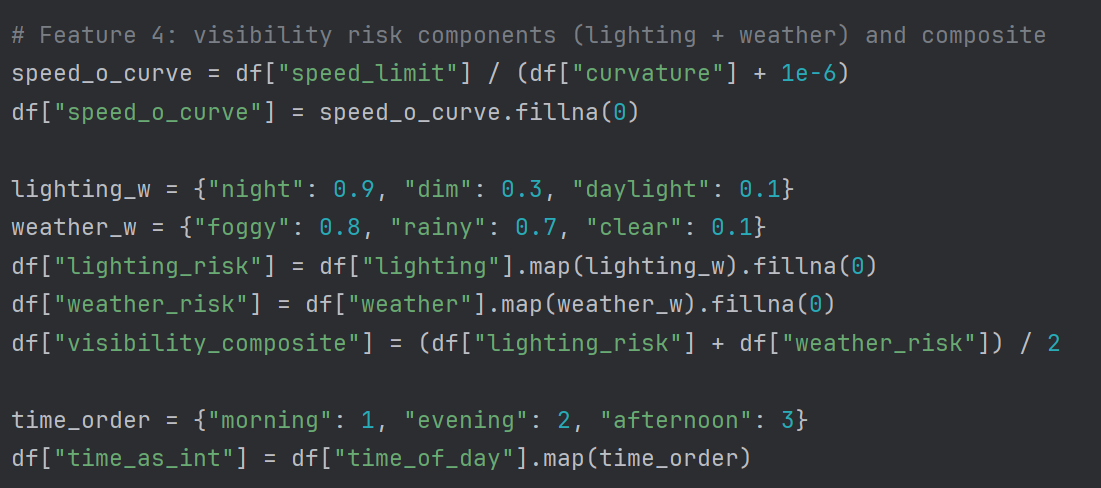
Jeg tester en og en interaksjon hver for seg og sammenligner med baseline.

Til slutt endte jeg opp med en liste på 12 engineered features som viste en score bedre enn baseline, med bruk av basefeatures + 1 engineered feature.

#### Grid search over features

Jeg satt opp et grid search over alle features for å se hva som interakterer godt sammen, og hva som kan være de nyttigste features å ta med videre.

Gjennom søket fant jeg at alle basefeatures + 5 engineered gir et godt resultat.



# **3: HYPERPARAMETERISERING**

Vi har vår baseline, og vi har feature sett. Nå ønsker jeg å finne best mulig parametre til å få en mest mulig nøyaktig modell.

Ut ifra Kaggle konkurransen tidligere i semesteret, hadde jeg god erfaring med Optuna til å søke etter gode parametere. Dette er en form for Bayesian optimization. [[6]](https://www.zotero.org/google-docs/?DzRB5m)

Jeg kjørte først flere litt brede søk for å finne noen gode utgangspunkt.

Etterpå kjørte jeg noen større finere søk for å forsøke å finne best mulig parametere.

Jeg endte opp med en RMSE på **0.05555**, på kaggles public leaderboard, med parameterne:



## Stacking

Videre ønsket jeg å finne flere gode modeller som kan stackes for å finne en enda bedre modell for å få en virkelig god modell.

**Dette ble nok litt nedprioritert.**

Vi lagde raskt en RandomForestRegressor[[4]](https://www.zotero.org/google-docs/?n4zkRY) og en ElasticNet [[7]](https://www.zotero.org/google-docs/?PiMlYR)modell, som vi dessverre ikke brukte like mye tid på, med samme features som LGBM og minimalt med hyperparametrisering

Deretter stacket vi det med sklearns StackingRegressor() [[8]](https://www.zotero.org/google-docs/?8KPzOd)

Resultatet ble også her **0.5555**, samme som LGBM for seg selv, på kaggles public leaderboard.

Hovedobservasjonen jeg ser er at dette reduserer ekstreme punkter i target “accident\_risk”. Det vil si de med høyest og lavest risiko går mot medianscore. Men at resultatet ikke ble merkbart bedre.

Vi antar resultatet kanskje er mer generaliserbart, og vil levere både stacked modell og LGBM modellen i konkurransen.

# 4: RESULTAT AV KONKURRANSEN

Som en ekstra del av oppgaven ønsker jeg å presentere hvordan vår modell gjør det i konkurranse med de andre.

## Kaggle leaderboard

Vi får utgitt et sett med testdata hvor vi skal predikere “accident\_risk” Vi lager en csv-fil med våre prediksjoner, som vi sender inn til Kaggle. De gir oss automatisk en RMSE score basert på 20 % av datasettet som blir et public leaderboard. Når konkurransen stenger vil vi se hvordan vi gjør det på de resterende 80 % av datasettet.

#### Public leaderboard

Vi har levert flere modeller

1. Alle engineered features lgbm modell uten hyperparametisering
   1. 0.05564
2. 5 engineered features lgbm modell uten hyperparametisering
   1. 0.05558
3. 5 engineered features lgbm modell medhyperparametisering
   1. 0.05555
4. 5 engineered features stacked modell medhyperparametisering
   1. 0.05555

#### Private leaderboard

1. 5 engineered features lgbm modell medhyperparametisering
   1. 0.05575
2. 5 engineered features stacked modell medhyperparametisering
   1. 0.05574

Dette gav oss leaderboard posisjonen **545 / 4083** som vi sier oss godt fornøyd med.

# 5: DEPLOYMENT

Etter at modellen var trent, eksporterte jeg den fra Python som en .joblib fil. Denne filen inneholder alt som trengs for å gjøre prediksjoner.

Målet var å gjøre modellen tilgjengelig via et nettsted, slik at hvem som helst kan skrive inn informasjon om en vei og få beregnet ulykkesrisiko med en gang. For å få dette til, koblet jeg modellen til webapplikasjonen ved hjelp av FastAPI

## Hvordan det fungerer

Hele systemet består av to hoveddeler som kommuniserer via HTTP:

1. Spring Boot webapp

Dette er et enkelt nettsted med en <form> der brukeren fyller inn informasjon om forholdene på veien. Når brukeren trykker på Predict, sender kontrolleren en POST-request til <http://localhost:8000/predict> og venter på svar. Når FastAPI sender tilbake resultatet, blir det gjort om til et lettleselig format og vist direkte på nettsiden.

2. FastAPI (Python-backend ligger i predict\_api.py)

Dette er et lite Python-API som kan motta og sende HTTP-requester i JSON format.

I dette tilfellet brukes det som koblingen mellom modellen og nettsiden. FastAPI lytter på port 8000 etter kall til endepunktet /predict. Når den mottar input-parametere i JSON, brukes de til å kjøre modellen. Resultatet konverteres deretter til JSON og sendes tilbake til webappen

## Oppsett på server

Ved en faktisk utrulling på web ville jeg satt opp en firewall som blokkerer direkte tilgang til portene 8000 og 8080 fra internett, men lar dem kommunisere internt. Deretter kunne jeg brukt Nginx eller Apache til å hondtere HTTPS (port 443) og videresende forespørsler til 8080.

## Monitorering og vedlikehold

For monitorering kunne jeg satt opp logging som lagrer alle forespørsler med parametre og returverdier, samt en service som varsler hvis noe uvanlig skjer i loggene.

Jeg kunne også laget en periodisk sjekk som sørger for at både modellen og fastAPI faktisk fungerer som de skal.

Når det gjelder vedlikehold, kan vi trene modellen på nytt med jevne mellomrom når nye ulykkesdata blir tilgjengelige. Siden systemet er delt i to komponenter kan jeg enkelt bytte ut .joblib filen uten å endre noe på nettsiden.

## Forbedringsmuligheter

En stor forbedring jeg ser for meg er å koble applikasjonen til Google Maps API slik at brukeren kan velge et område de skal kjøre i, i stedet for å fylle ut parametere manuelt. Applikasjonen kunne da automatisk finne de relevante veiparametrene basert på valgt område. Hvis området er stort, kunne det deles opp i mindre soner, og risikoen kunne beregnes som et gjennomsnitt av alle disse. Dette ville gjort løsningen mer brukervennlig og relevant for virkelige kjøresituasjoner.

# 6: KODE / VIDEO

Koden er offentlig tilgjengelig: <https://www.kaggle.com/code/terjem/lgbm-rf-enet-5-features-0-5574-private-lb>

Videoen: <https://www.youtube.com/watch?v=lxVvuebNyr0>

# 7**: REFERANSER**

[[1]](https://www.zotero.org/google-docs/?yj3sFl) [«Predicting Road Accident Risk». Åpnet: 24. oktober 2025. [Online]. Tilgjengelig på: https://kaggle.com/playground-series-s5e10](https://www.zotero.org/google-docs/?yj3sFl)

[[2]](https://www.zotero.org/google-docs/?yj3sFl) [«root\_mean\_squared\_error», scikit-learn. Åpnet: 25. oktober 2025. [Online]. Tilgjengelig på: https://scikit-learn/stable/modules/generated/sklearn.metrics.root\_mean\_squared\_error.html](https://www.zotero.org/google-docs/?yj3sFl)

[[3]](https://www.zotero.org/google-docs/?yj3sFl) [«StandardScaler», scikit-learn. Åpnet: 1. november 2025. [Online]. Tilgjengelig på: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html](https://www.zotero.org/google-docs/?yj3sFl)

[[4]](https://www.zotero.org/google-docs/?yj3sFl) [«RandomForestClassifier», scikit-learn. Åpnet: 26. oktober 2025. [Online]. Tilgjengelig på: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html](https://www.zotero.org/google-docs/?yj3sFl)

[[5]](https://www.zotero.org/google-docs/?yj3sFl) [«OneHotEncoder», scikit-learn. Åpnet: 1. november 2025. [Online]. Tilgjengelig på: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html](https://www.zotero.org/google-docs/?yj3sFl)

[[6]](https://www.zotero.org/google-docs/?yj3sFl) [«Optuna: A hyperparameter optimization framework — Optuna 4.5.0 documentation». Åpnet: 1. november 2025. [Online]. Tilgjengelig på: https://optuna.readthedocs.io/en/stable/index.html](https://www.zotero.org/google-docs/?yj3sFl)

[[7]](https://www.zotero.org/google-docs/?yj3sFl) [«ElasticNet», scikit-learn. Åpnet: 1. november 2025. [Online]. Tilgjengelig på: https://scikit-learn/stable/modules/generated/sklearn.linear\_model.ElasticNet.html](https://www.zotero.org/google-docs/?yj3sFl)

[[8]](https://www.zotero.org/google-docs/?yj3sFl) [«StackingRegressor», scikit-learn. Åpnet: 1. november 2025. [Online]. Tilgjengelig på: https://scikit-learn/stable/modules/generated/sklearn.ensemble.StackingRegressor.html](https://www.zotero.org/google-docs/?yj3sFl)