Digit recognition with MNIST

Using supervised machine learning

Keikiet Pham

EC Utbildning

Inlämningsuppgift 2

202403

Abstract

This report reflects the process of building and deploying a supervised machine learning model for digit recognition using the MNIST dataset and demonstrates the process of model development, including hyperparameter tuning and deployment though a Streamlite application. The researcher then examines in whether there are differences in prediction accuracy between model testing and deployment and explores methods to enhance the performance of the deployed model.

Table of contents

1 Introduction

In today's data-driven world, machine learning (ML) knowledge is becoming an increasingly essential skillset intertwined with the field of data science and together can generate incredible values to organizations whether it is extraction of meaningful patterns and predictions from data, automating tasks & processes or empowering innovation. According to an article written and published by Preston Fore, Fortune.com January 25, 2024. One million machine learning specialists are needed by 2027.

In this thesis report we will demonstrate how one with no prior ML experience after a 5-week ML course can create basic ML models using scikit-learn trained on the MNIST dataset to predict numbers from an image.

1.1 Aim and research questions

The purpose of this report is to demonstrate how a student who has just started learning ML can create a supervised ML model, train it on the MNIST dataset and use the same model to predict a handwritten digit uploaded via a Streamlit app created by the student.

- Will the prediction accuracy differ from the model's testing phase and the deployment output from the Streamlit application, and if so, what factors contribute to these differences?

- Based on the results obtained, what conclusions can be drawn and potentially what more could have been done to improve potential results?

2 Theory

There are many different types of machine learning techniques that can be categorized in Supervised, Unsupervised, semi-supervised and Reinforcement learning. For this project where we will predict a target numeric digit (classification task) and apply supervised learning techniques to do so.

By feeding our models examples of handwritten digits (training set) from MNIST dataset including both predictors and their labels, the model algorithm will learn the pattern and relationships between pixels (features) to predict a digit from an entirely new image. In theory the performance of the model will depend on what algorithm is used, how it's tuned and the amount of data we train the model with. Generally, the more data is fed to the models the better they learn and predict on new data. This section presents theory which is relevant to understanding the context of this project.

2.1 Supervised Learning algorithms pros and cons

In this section we will look at 4 different model algorithms for classification tasks and highlight the pros and cons for each model (Scikit-learn.org)

2.1.1 RandomForestClassifier

Random forest is a popular supervised machine learning algorithm and can be used for solving regression and classification problems. A Random Forest classification is a collection of multiple decision trees, all generated and using different subsets of the data and features (sampling with replacement). Each decision tree is like an expert, providing its opinion on how to classify the data. Random forest is a bagging algorithm (Datacamp, 2023).

Strengths:

- Can handle linear and non-linear relationships well.
- Generally, provides high accuracy and balances the bias variance trade-off well.
- Can be used for both classification and regression tasks.

Weaknesses:

- Can be computationally intensive for large datasets.
- Not easily interpretable with very little control over what the model does.

2.1.2 KNearestNeighbors

The KNN algorithm can be considered a voting system, where the majority class label determines the class label of a new data point among its nearest "k" neighbors (Brilliant.org).

Strengths:

- Easy to understand and implement.
- Does not assume any probability distributions on the input data.
- Can quicky respond to changes in input.

Weaknesses:

- Sensitive to localized data.
- Can be computationally intensive for large datasets.
- Sensitive to dataset imbalance.

2.1.3 Support Vector Machines (SVM)

SVM often offers very high accuracy with the right hyperparameter tuning, it's strength comes from the multiple kernel options which all has their own strengths and weakness. SVMs main disadvantage is that it is not suitable for large datasets because of its demanding training time (Datacamp, Dec 2019).

Strengths:

- Powerful and versatile, capable of performing linear, nonliear classification, regression and even outlier detection.
- Many hyperparameter kernels that are easy to implement and can add versatility.

Weaknesses:

- Sensitive to feature scales.
- Unlike Logistic Regression classifiers, SVM classifiers do not output probabilities for each class.
- Computationally intensive.

2.1.4 Stochastic Gradient Descent Classifier

SGD is a fairly simple yet very efficient version of gradient descent algorithm, especially for its training speed. Instead of considering the full batch gradient on all training data points SGD picks a training data point at random and then only considers the error on that data point (Abu-Mostafa, Magdon-Ismail, Lin, 2012, p. 97).

Strengths:

- Efficient especially and large datasets.
- Easy to implement (lots of tuning opportunities).

Weaknesses:

- Requires a relatively high number of hyperparameters such as regularization parameter and the number of iterations.
- Sensitivity to feature scaling, ideally standardized scaling.

2.1.5 VotingClassifier

Voting classifiers is an ensemble learning method that aggregates the predictions of each classifier and then either predicts the class that gets most votes (hard voting classifier) or predicts the class with the highest-class probability, averaged over all the individual classifiers (soft voting). Because of this, Voting Classifiers often achieves a higher accuracy than the best classifier in the ensemble. Soft voting is often considered a stronger a higher performer than hard voting because it gives more weight to highly confident votes (Géron, 2019, p189-190).

Strengths:

- Often reduces overfitting and improves generalization.
- Robust to noisy data and outliers.

Weaknesses:

- Can be computationally intensive if they involve a larger number of base models or require extensive tuning of hyperparameters.
- Can be hard to interpret if they involve many larger number of base models.

2.2 Feature scaling

The two most popular types of feature scalers are Standardscaler and MinMaxScaler.

The former works via standardizing the data (i.e. centering them) meaning bringing Mean value = 0 and standard deviation = 1. It gets affected by outliers and works best when used when the dataset has a Gaussian-Like Distribution.

MinMaxScaler is preferred when we want to bring our features into a specific range. This method gets heavily affected by outliers (Géron, 2019, p69-70).

2.3 Overfitting

Overfitting occurs when an algorithm fits too closely or even exactly to its training data, resulting in a model's inability to predict accurately or draw conclusions from any new data other than the training data defeating the purpose of a machine learning model. This typically happens when the model trains for too long on the sample data or when the model is too complex, resulting in the model memorizing the "noise" or irrelevant information in data. Low error rates combined with a high variance are good indicators of overfitting. Another way of identifying overfitting is to set aside part of the training dataset as a "test/validation set", if the training data has a significantly lower error rate and test data has high error rate, it signals overfitting. ("IBM," n.d).

Possible solutions in dealing with overfitting is getting more data, data cleaning, reduce noise in the training data and simplifying the model (reduce number features or parameters used, regularizing the model and/or selecting a simpler algorithm). Using an ensemble learning algorithm can also reduce overfitting. (Géron, 2019, p720).

2.3.1 Cross-Validation

Instead of splitting the dataset into smaller training and a validation set and shrinking the training data an alternative is to do a Cross-Validation where the training set is split into "k"-subsets called folds. One of the folds then serves as a validation set, the remaining folds evaluates against. This process is iterated k- times changing the validation fold in each iteration. (Géron, 2019, p73).

2.3.2 Hyperparameter tuning (HPs-T)

Hyperparameters tuning is the process when a model's configuration variables are tweaked to control the model structure, function and performance for optimal results. Selecting the right set of hyperparameters is difficult but nevertheless important to improve model performance, accuracy and avoid over- & underfitting. Two common HPs-T techniques are Grid search and Random Search ("amazon web services," n.d).

3 Methodology

This section is divided into two subheaders amd demonstrate what the different steps of the project.

In 3.1, a complete Machine Learning-process flow end to end is demonstrated and in 3.2 process steps in building Streamlit app is demonstrated.

3.1 Model Design

All coding will be done in Jupyter notebook and then the final model be saved in a local file using joblib dump.

3.1.1 Data Collection

The data used for the project is MNIST_784 (Modified National Institute of Standards and Technology) and consists of 70 000 handwritten digits. Imported through scikit-learn's function: fetch_openml. All images have the sizes of (28 x 28 = 784 pixels)

The data is then assigned to variable X and y, depending on whether it is features or labels. Next, we split the dataset consisting of 70 000 rows into a train, validation and test set.

The idea is to have a large training set to train models, a smaller validation set to evaluate hyperparameter tuning and lastly a test set to see how well our final model performs on new data.

```
Training Features: (42000, 784)
Training Labels: (42000,)
Validation Features: (14000, 784)
Validation Labels: (14000,)
Test Features: (14000, 784)
Test Features: (14000,)
```

3.1.2 Data Visualization

Next, we explore the training dataset to get to know the data. More specifically we want to investigate Label distribution and get a glance of the handwritten digits.

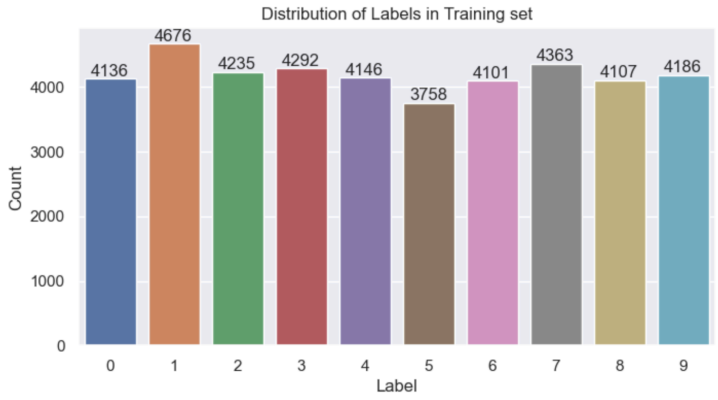


Figure 1. Label Distribution of 10 classes in training set

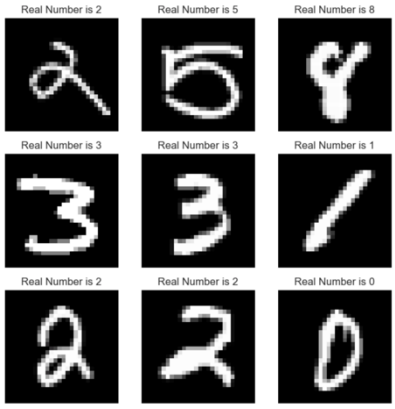


Figure 2. Handwritten digits from training set

3.1.3 Data Preprocessing & Feature Engineering

Now that we have a better understanding of the data, we know the label distribution is even (balanced dataset), we know the images consists of pixels with density ranging from [0, 255] and we see that the handwritten digits correspond realistically with their labels – we assume this is true for the whole dataset. We can also see by printing out the 5 rows of X_train that the pixel intensities most likely are inverted, meaning black pixels have pixel values 0 and whiter pixels has values closer to 255.

Next, we scale features to make them all the same magnitude (i.e. importance or weight).

This can be done by either standardizing or normalizing features. Since we have discovered that our dataset is balanced (i.e. not Gaussian-Like Distributed) we will use MinMaxScaler to bring our features into range [0, 1].

3.1.4 Algorithm Selection

So how do we know what estimator and algorithm to choose for our final model when there are so many to choose from?

As this can be very difficult to say without trying different algorithms we will select and train a few models.

Due to computational constraint, the idea is to train 4 base models without tweaking (no HPs-T) with k-fold validation (k = 5) on the **training set** evaluate and pick top 1-2 performer to hyperparameter tune with GridSearchCV on the training set, predicting only on **Validation set**. Finally, we implement a Voting Classifier with soft voting for our final model and save it locally using joblib dump.

Based on scikit-learn algorithm cheat-sheet (“scikit-learn.org”, n.d) we pick our 4 base models.

1. RandomForestClassifier
2. KNeighborsClassifier
3. LinearSVC
4. SGDClassifier

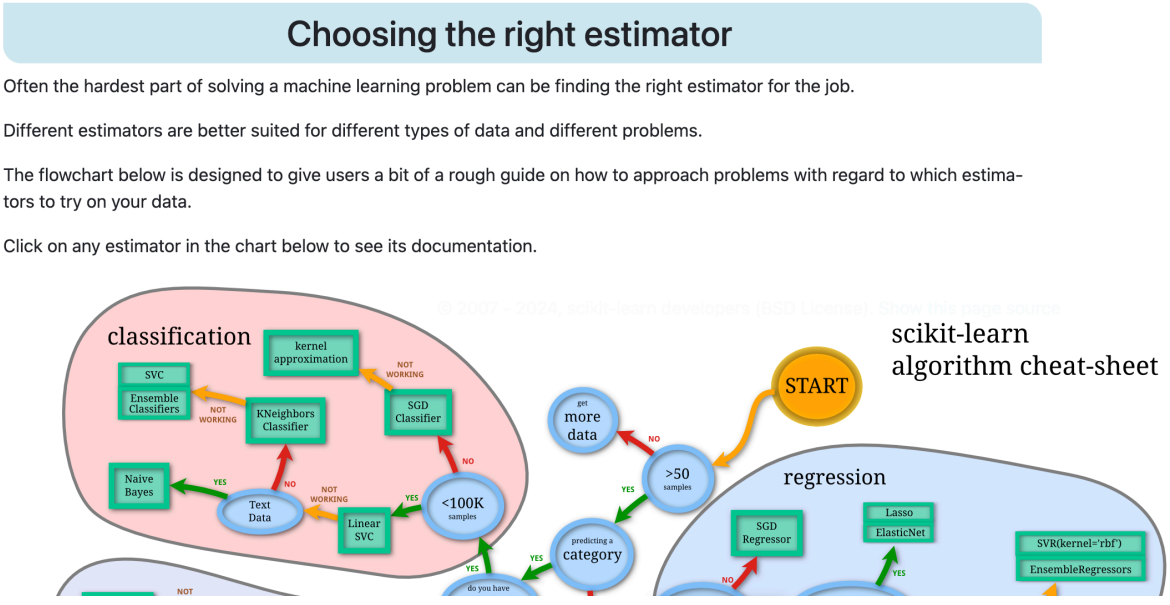


Figure 3. Snippet of Scikit-learn algorithm cheat-sheet focusing on classification problem. See full cheat-sheet in appendix.A

3.1.5 Model training steps and deployment (Save model in a local file)

Step 1: Initialize and Train Models:

- Initialize four different classifiers: Random Forest, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Stochastic Gradient Descent (SGD) Classifier.
- Train each model using the training data (`X_train` and `y_train`) separately.
- Perform k-fold cross-validation (with k=5) on each model to evaluate their performance. (since Random Forest and KNN models perform noticibly better we will continue these models to hyperparameter tune).

Step 2: Hyperparameter Tuning for RandomForestClassifier & KNeighborsClassifier:

- Define a grid of hyperparameters for the Random Forest classifier ('n_estimators', 'max_depth', 'min_samples_split').
- Define a grid of hyperparameters for the KNeighborsClassifier ('n_neighbors', 'weights', 'p') `min_samples_split`).
- Use GridSearchCV to search for the best combination of hyperparameters that optimizes accuracy on the training data.
- Fit the GridSearchCV (CV=5) to the training data (`X_train` and `y_train`) and predict on X_Val (validation set) to find out best accuracy score achieved during hyperparameter tuning.

Step 3: Initialize and train a soft Voting Classifier on training set combining the tuned Random Forest & KNN models and predict on X_val to see performance on new data and if model is overfitting.

- Retrain the model on full training set (X_train + X_val) and predict on y_test to see final score.
- Save the model in a local file.

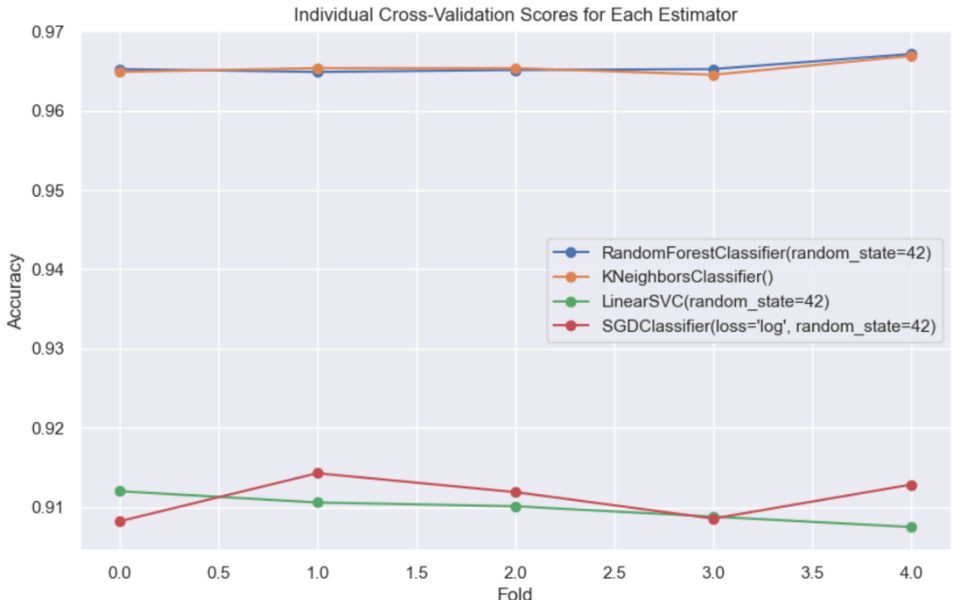


Figure 4. Accuracy score from 5-fold validation performed on the initial 4 base models.

3.2 Streamlit app

First, we define the app, its features and constraints. The idea is to create an app with the feature of uploading an image, preprocess the image and using our imported model to predict the digit on the uploaded image. For this we will import following packages: skimage, streamlit, numpy, PIL and joblib. Additionally, we will import matplotlib to visualize preprocessed image and histograms.

3.2.1 Data preparation and image preprocessing

So now that we have imported the required packages, our model and uploaded an image. We now need to preprocess the image so that our model can finally predict the digit. The main purpose of image preprocessing is to transform the uploaded image so that its format and features is like the MNIST digit images we trained our model on. As a reminder, we already concluded (see 3.1.3 Data Preprocessing & Feature Engineering) that all images from MNIST have inverted colors and has the size of 28 x 28 pixels.

In the first image preprocessing step, we convert the uploaded image to a gray scale if it has more than 3 channels (indicating the image is RGB colored). We then invert the gray scaled image and resize the image to 28 x 28 format. Lastly, we apply filter threshold to remove any noises and enhance the distinction between foreground (digit) and the background. Now that image preprocessing is done, we can predict the digit in the uploaded image.

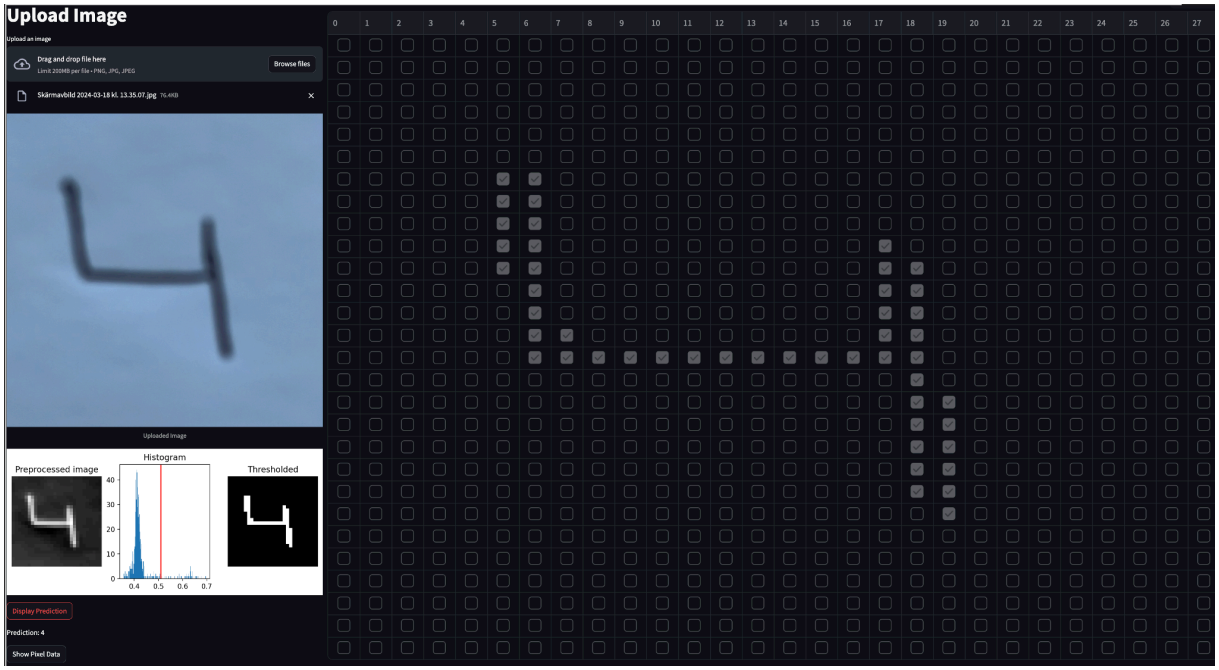


Figure 5. Streamlit app, Left: displaying Uploaded image, preprocessed, pixel-intensity histogram and thresholded image. Right: illustrating 28x 28 pixels after thresholding. Clustering foreground (digit) and background. Predicted digit is: 4

4 Results & discussions

4.1 Prediction model

So, our final model trained on MNIST dataset is a Voting Classifier consisting of Random Forest & KNN model. Despite hyperparameter tuning, the impact on the model accuracy performance was almost unnoticed. Two main factors may have contributed to this observation.

Firstly, the hyperparameter tuning process might have been constrained by a narrow or insufficiently deep search space. In this project, the hyperparameter search space was limited, potentially missing out on optimal configurations. Exploring with additional parameters and widen ranges within the parameter could potentially lead to more significant improvements in model performance.

Secondly, data quality and complexity of the dataset may have an impact. If the dataset is simple and lack complexity, default parameters may perform satisfactory results without being tweaked.

Despite these challenges, the final model demonstrated good performance, with precision, recall and F1-score indicating robust performance across various metrics.

4.2 Streamlit app

The results obtained from predicting digits from uploaded images via the Streamlit demo app revealed interesting insights. Noticeably, prediction accuracy seemed to vary depending and reliant on certain conditions and characteristics of the uploaded images.

Observations suggests that the prediction accuracy was notably higher for images with certain characteristics, such as enlarged and centered digits against a uniform background. This points towards the importance of image preprocessing techniques to improve prediction accuracy.

4.3 Opportunities for improvement and future work

While the results we've achieved in the demo look promising under right conditions there are still things to consider and opportunities for further investigation. Firstly, we could do a more thorough testing of the Streamlit app by trying out with lots of different kinds of images the help understand how well the model works in different conditions.

Additionally, we could find better ways to preprocess the images like adding features to handle different lighting, rotations and offsets from the center.

Lastly, we can investigate different deep learning techniques and improve our algorithm.

		X_train	predict X_Val	predict X_test
	Tuned	Accuracy **(bold=mean)**	Accuracy **(bold=mean)**	Accuracy **(bold=mean)**
RandomForestClassifier		**0.966**	0.965	
KNeighborsClassifier		**0.965**	0.969	
LinearSVC		**0.910**		
SGDClassifier		**0.911**		
RandomForestClassifier	x		**0.967**	
KNeighborsClassifier	x		**0.973**	
VotingClassifier	x		0.975	0.975

Figure 5 Model accuracy

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1343
1	0.97	0.99	0.98	1600
2	0.98	0.97	0.97	1380
3	0.98	0.96	0.97	1433
4	0.98	0.97	0.97	1295
5	0.97	0.97	0.97	1273
6	0.98	0.99	0.99	1396
7	0.97	0.98	0.97	1503
8	0.99	0.95	0.97	1357
9	0.96	0.96	0.96	1420
accuracy			0.97	14000
macro avg	0.98	0.97	0.97	14000
weighted avg	0.97	0.97	0.97	14000

Figure 6. Voting Classifier Classification Report

5 Conclusion

This project aimed to address two key research questions:

Model Testing vs. Deployment: We investigated whether there are differences in prediction accuracy between testing the model and deploying it via the Streamlit app. Our findings revealed that indeed, there are big variations in accuracy. This highlights the importance of thorough evaluation and adjustments during deployment to ensure consistent performance in real-world settings. Probably the biggest difference in our case comes from insufficient image preprocessing to cover different characteristics of the image we try to predict.

Effectiveness of the Deployed Model and Improvement steps: Based on our results, we concluded that the deployed model showed promise, but there's room for improvement. We identified image preprocessing as a crucial factor influencing prediction accuracy. By focusing on enhancing preprocessing techniques, such as resizing and normalization, we can potentially boost the model's performance.

6 Teoretiska frågor (swedish)

Besvara nedanstående teoretiska frågor koncist.

1. Kalle delar upp sin data i ”Träning”, ”Validering” och ”Test”, vad används respektive del för?

> **Träningsdata** används för att träna utvalda modeller, V**alideringsdata** används för att utvärdera och välja ut modeller att gå vidare med, **Testdata** används för att testa hur bra den utvalda finjusterade modellen predikterar ny data.

2. Julia delar upp sin data i träning och test. På träningsdatan så tränar hon tre modeller; ”Linjär Regression”, ”Lasso regression” och en ”Random Forest modell”. Hur skall hon välja vilken av de tre modellerna hon skall fortsätta använda när hon inte skapat ett explicit ”validerings-dataset”?

> Implementera cross-validation på samltliga modeller för att få genomsnittlig scoring-resultat och jämför därefter resultatet mellan modellerna.

3. Vad är ”regressionsproblem? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden?

> *Regressionsproblem är ett supervised learning problem där den oberoende variabeln (y) antar kontinuerliga värden. Exempel på modeller: Linear Regression, Support Vector Machines (SVM) och DecisionTrees.*

4. Hur kan du tolka RMSE och vad används det till:

> *Root Mean Squared Error är ett scoring-resultat och är avståndet mellan modellers prediktioner och de sanna värdena. RMSE används bl.a. i Regressionsproblem.*

5. Vad är ”klassificieringsproblem? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden? Vad är en ”Confusion Matrix”?

> Klassificeringsproblem är likt Regressionsproblem också ett supervised learning problem, däremot antar den oberoende variablen (y) diskreta värden. Tillämpningsområden kan vara att identifiera olika objekt i en bild eller klassificering av kunder i en kundbas. Exempel på modeller är KNN, SVM och RandomForest.
>
> Confusion Matrix är Matristabell med två dimensioner (”actual” och ”predicted”) och används för att visualisera och utvärdera en modells (algoritmens) prestation.

6. Vad är K-means modellen för något? Ge ett exempel på vad det kan tillämpas på.

> *K-means är en Unsupervised maskininlärningsmodell som är kapabel att klustra data snabbt och effektivt, ofta med endast med ett par iterationer och kan används vid t.ex. bildsegmentering, dimensionsreducering vid preprocessing samt Semi-Supervised Learning.*

7. Förklara (gärna med ett exempel): Ordinal encoding, one-hot encoding, dummy variable encoding. Se mappen ”l8” på GitHub om du behöver repetition.

> Samtliga ovanstående encodings är en preprocessing tekniker som används för att konvertera kategoriska data till numeriska värden så att maskinlärningsmodellera kan avläsa och lära sig av informationen.

Ordinal encoding: Används när den kategoriska datan kan rankas. Tex. ”storleks” variabel med värden [”liten”, ”mellan”, ”stor”] kan då konverteras till [0, 1, 2].

One-hot encoding: Används när kategoriska data inte kan rankas. Varje feature eller egenskap antar värden [0, 1] och bildar en ny kolumn.

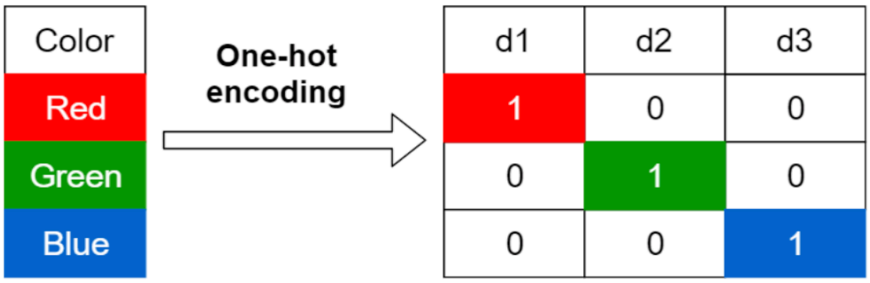


Figure 7. illustrating One-hot encoding

Dummy variable encoding: Används likt One-hot encoding när kategoriska data inte kan rankas. Skillnaden är att man nyttjar uteslutningsmetoden i den instans där samtliga kolumner innehåller [0, 0, …, 0] och därmed kan utesluta en kolumn, man undviker på så vis onödig redundans i.

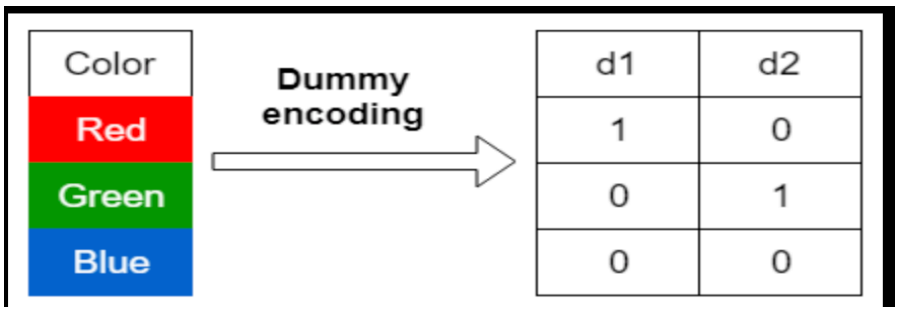


Figure 8. illustrating Dummy encoding

8. Göran påstår att datan antingen är ”ordinal” eller ”nominal”. Julia säger att detta måste tolkas. Hon ger ett exempel med att färger såsom {röd, grön, blå} generellt sett inte har någon inbördes ordning (nominal) men om du har en röd skjorta så är du vackrast på festen (ordinal) – vem har rätt?

Julia har rätt att det är en tolkningsfråga och det hela beror på om den ”röda” skjortan har ett högre värde och signifikans än tex grön och blå. Man kan tänka sig en termometer som visar rött när det är väldigt varmt, grönt när det är ”lagom” varmt och blått när det är kallt, då skulle man kunna rangordna färgerna efter hur hög temperaturen är.

9. Kolla följande video om Streamlit: https://www.youtube.com/watch?v=ggDa-RzPP7A&list=PLgzaMbMPEHEx9Als3F3sKKXexWnyEKH45&index=12
Och besvara följande fråga:
- Vad är Streamlit för något och vad kan det användas till?

Streamlit är ett gratis program med öppen källkod som möjliggör snabb generering och visualisering av till exempel maskininlärnings modeller i en webapp.

7 Självutvärdering (swedish)

1. Utmaningar du haft under arbetet samt hur du hanterat dem.

Mina största utmaningar har varit dels att komma i gång med Streamlit appen, insåg nog inte riktigt att det var paket/bibliotek likt matplotlib man skulle importera och att man kunda koda precis som vanligt. Löste det genom att läsa på nätet och kolla andra tutorials utöver din.

En annan utmaning var rapportskrivandet, det var längesedan man gjorde något liknande och det kändes verkligen som att man var rostig men det blev lättare efterhand trots att det tog mycket längre tid än räknat särskilt när rapporten skrevs på engelska vilket var lärorikt.

2. Vilket betyg du anser att du skall ha och varför.

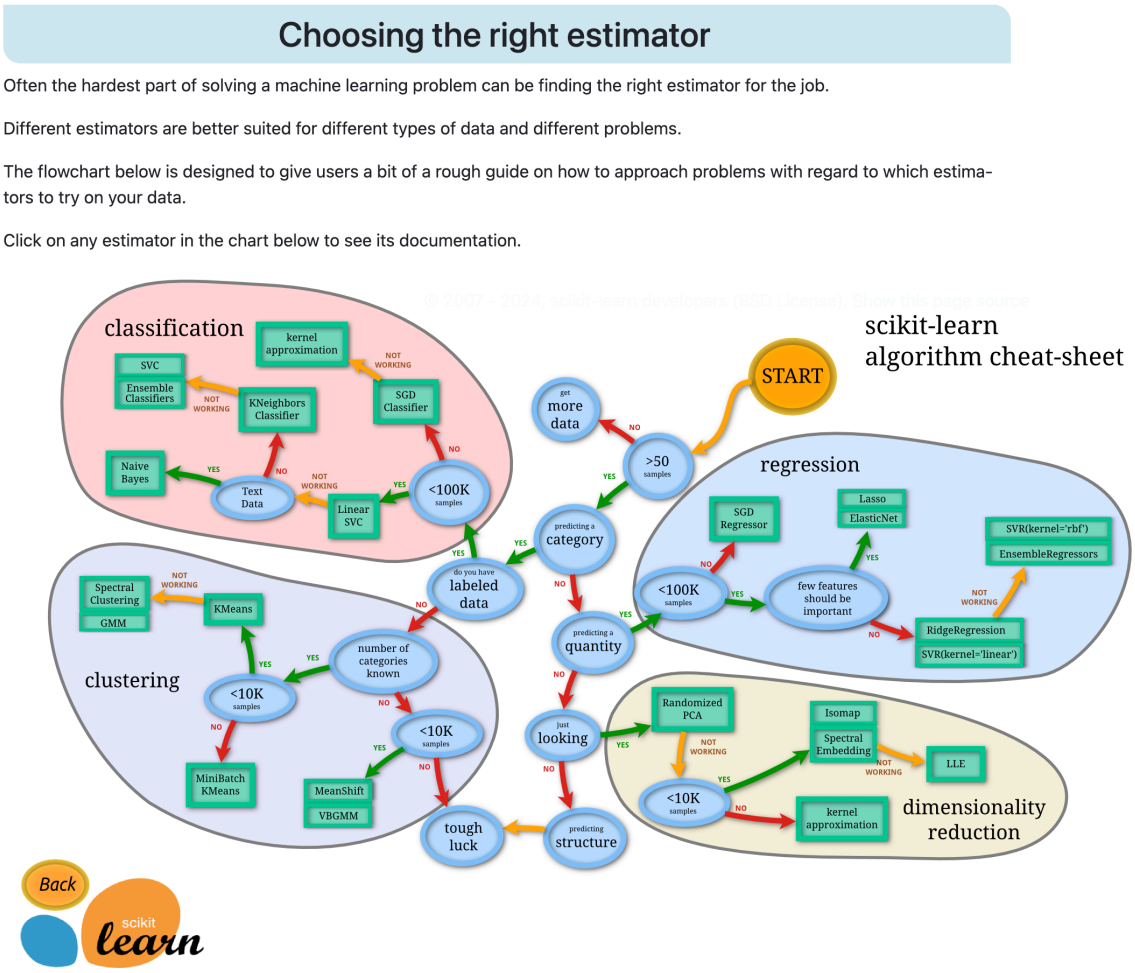
VG. Jag hoppas att jag kunnat framhäva min förståelse om maskininlärning i rapporten men också tillämpningar i min modell. Det var nyttigt att skriva rapporten då man tvingas göra faktakontroller men också reflektera och formulera meningar i rapporten.

3. Något du vill lyfta fram till Antonio?

En rolig inlämning men hade gärna föredragit mer fokus på maskininlärning som knyter ihop det vi lärt oss under kursens gång. Rapportskrivningen tar mycket tid men det är lärorikt. Man lär sig mycket om ämnet av att skriva rapport.

Appendix A

Figure 9. Scikit-learn algorithm cheat-sheet



Source list

Géron, A. (2019. In: O´REILLY. *Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow (2nd ed.)* Canada.

Abu-Mostafa, Magdon-Ismail, Lin. (2012). In: *Learning From Data, a short course* (p. 97). AMLbook.

Datacamp. (Dec 2019) *Support Vector Machines with Scikit-learn Tutorial.* https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

Datacamp. (Feb 2023) *K-Nearest Neighbors (KNN) Classification with scikit-learn.* https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

Datacamp. (Feb 2023) *Random Forest Classification with Scikit-Learn.* https://www.datacamp.com/tutorial/random-forests-classifier-python

Amazon Web Services.com. (n.d) *What is hyperparameter tuning.* https://aws.amazon.com/what-is/hyperparameter-tuning/#:~:text=Hyperparameters%20directly%20control%20model%20structure,values%20is%20crucial%20for%20success.

IBM. (n.d) *What is overfitting?* https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn