JÄMFÖRELSE AV MASKININLÄRNINGSMODELLER FÖR HANDSKRIVEN SIFFERKLASSIFICERING

Matthew H. Motallebipour

5 april 2024

Yrkeshögskolan EC Utbildning

Sammanfattning

Abstract: The present report elaborates on a comparison between three different machine learning models for classification of handwritten digits: Support Vector Machines (SVM), Random Forests (RF), and Extreme Gradient Boosting (XGBoost). The models are compared and ranked based on accuracy. In the second part of the report, an ensemble learning model is also presented, which is used to create a Python application in the Streamlit framework that classifies uploaded images with an accuracy of about 98%.

Sammanfattning

Abstract: Föreliggande rapport presenterar en jämförelse mellan tre olika maskininlärningsmodeller för klassificering av handskrivna siffror: Support Vector Machines, Random Forest, och Extreme Gradient Boosting. Modellerna jämförs och rangordnas med avseende på noggrannhet. I den andra delen av rapporten presenteras också en ensemble inlärning modell som används för att framställa en Python-applikation i ramverket Streamlit som via en server klassificerar uppladdade bilder med en noggrannhet på cirka 98%.

Acknowledgements

I would like to thank my supervisor, Mr. Antonio Prgomet, for his outstanding teaching and his guidance throughout the whole project. I would also like to thank my dear friend, Mr. Robert Shaw for his invaluable contributions; a peer through whose journalistic scrutiny I was able to realize the shortcomings in my work and improve upon them. Finally, thank you to all my fellow students that are always there for a good discussion throughout all courses.

Förkortningar

SVM	Support Vector Machines
RF	Random Forest
XGBoost	Extreme Gradient Boosting
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
GUI	Graphical User Interface
API	Application Programming Interface
IDE	Integrated Development Environment

Figurer

Tabeller

Innehåll

1 Inledning

För bara några år tillbaka i tiden kunde man inte föreställa sig att algoritmer inom det nytillkomna heta området maskininlärning skulle på så kort tid revolutionera takniken och människors syn på framtiden. Speciellt efter introduktionen av djupinlärning (DL) och faltningsneuronnät (CNN) har maskininlärning (ML) blivit så populär att skolor och universitet runtom Sverige har börjat erbjuda kurser inom området för lärare för att kunna bemöta den ökande efterfrågan på ML-kunnandet (Skolverket, 2020). Därför är det ytterst väsentligt att vi som studenter och forskare inom området maskininlärning förstår de olika modellerna och deras styrkor och svagheter.

1.1 Syfte

Denna rapport ämnar behandla tre olika ML-modeller, nämligen Support Vector Machines (SVM), Random Forest (RF), och Extreme Gradient Boosting (XGBoost). Syftet är att besvara frågan om dessa modeller i allmännhet kan klassificera handskrivna siffror med en acceptabel noggrannhet. I synnerhet kommer vi att jämföra skillnaden mellan dessa tre och se vilken som fungerar bäst.

1.2 Frågeställning

Den specifika frågan är vilken av de tre modellerna SVM, RF och XGBoost är det bästa alternativet för klassificering av handskrivna siffror.

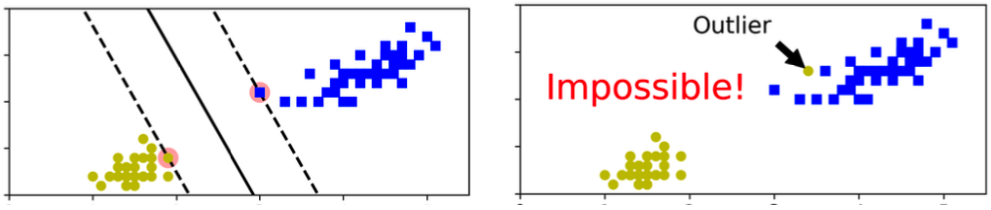
1.3 Textens disposition

Först kommer vi att titta på våra tidigare nämnda modeller och vad de är. Därefter presenteras det lite olika metoder för att förbehandla och senare klassificeras bilderna som används som träningsdata. I resultatdelen kommer vi att presentera hur våra modeller klarat av utmaningen. I diskussionsdelen kommer vi att jämföra och rangordna modellerna och i den sista delen kommer vi presentera vår slutsats och förslag på framtida arbete för att förbättra resultaten.

2 Teori

För klassificering av data i olika former såsom bild, text eller ljud finns det många olika alternativ inom maskininlärning. Dessa alternativ kallas modeller varav de mest kända är Logsitic Regression, Decision Trees, Random Forest, Gradient Boosting Machines, Support Vector Machines, k-Nearest Neighbors, Naive Bayes, Neural Networks, Linear Discriminant Analysis, Adaboost, Bootstrap Aggregating (Bagging), extra trees (Géron, 2019), Quadratic Discriminant Analysis (Bishop, 2006), Gaussian Process Classifiers (Gibbs and MacKay, 1997), Passive Aggressive Classifiers (Crammer and Singer, 2003) och många fler.

Vi valde att begränsa oss till en jämförelse mellan tre av dessa modeller: Support Vector Machines, Random Forest och Extreme Gradient Boosting. Dessa modeller är kända för att vara effektiva och används ofta i praktiken.



(a) (b)

Figur 1: Support Vector Machines: [Géron] (a) Linjär separerbar data. De två streckade linerna visar marginalen medan den hela linjen är beslutsgränsen (b) Icke-linjär separerbar data.

2.1 Support Vector Machines (SVM)

SVM är en linjär klassificerare som försöker hitta den bredaste marginalen som kan separera två klasser. Den används både för klassificering och regression men används mest för klassificering. SVM är en övervakad algoritm baserad på Vpnik-Chervonenkis teori (Duda et al., 2000). Om klasserna inte är linjärt separerbara

2.2 Random Forest (RF)

Random Forest är en ensemble inlärningsmodell som består av flera beslutsträd. Varje träd i skogen är en enkel klassificerare som bygger på slumpmässigt valda egenskaper/ kolumner i befinlig data. RF är också en mycket effektiv algoritm som tvärtemot SVM är robust mot outliers. Det är därför som vi har inkluderat den i implementeringen av vår sifferklassificerare. RF är med andra ord aggregerad modell vars utfall är baserad på en majoritsröstning av alla träden.

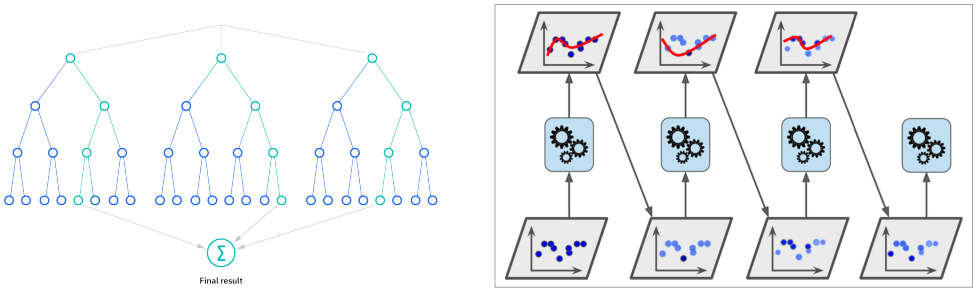
Leo Breiman, som förresten också är känd för att ha introducerat Bootstrap Aggregating (Bagging) (Géron, 2019), står bakom Denna numera mycket välkända modell, tillsammans med Adele Cutler (Izenman, 2008).

2.3 Extreme Gradient Boosting (XGBoost)

XGBoost är inte från samma katalog som de tidigare nämnda modellerna, utan den har en egen katalog i Python. XGBoost är en optimerad version av Gradient Boosting Machines some är en ensemble inlärningsmodell som liksom RF också består av flera beslutsträd. Skillnaden är att denna består av träd som byggs i en sekventiell ordning och inte parallellt som i RF. Varje träd i XGBoost bygger på felet, de så kallade residualer som är skillanden mellan predikterat och reelt värde, från det föregående trädet. Denna modell använder också early stopping för att undvika överanpassning till data och därmed minska bias (Géron, 2019). Early stopping innebhär helt enkelt att det enskilda trädet inte byggs på när residualen inte längre minskar avsevärt. XGBoost har visat sig vara snabbare än sin föregångare (Guido, 2016) och används därför ofta inom ML-tävlingar.

2.4 PCA

??? Add some text about PCA, EVR and such ...



(a) (b)

Figur 2: Random Forest (IBM, 2021) (a) och Gradient Boosting (Géron, 2019) (b)

3 Metod

För att jämföra de tre modellerna SVM, RF och XGBoost använde vi oss av Python-biblioteken Scikit-learn och XGBoost. All kod implementerades i Jupyter Notebook inbäddad i VSCode och på en Mac dator med 1.4 GHz Quad-Core Intel Core i5, 8 GB 2133 MHz LPDDR3, och Intel Iris Plus Graphics 645 1536 MB.

Nu ska vi titta närmare på vad vilka steg vi tog för att komma till mål.

3.1 Undersökning

Det första man brukar göra i alla projekt inom ML och data science är att titta på data och se hur den ser ut, vilka egenskaper den har samt vad som kan vara av värde att notera och använda.

Vår givna data bestod av MNIST-datasetet som innehåller bilder på 70 000 handskrivna siffror i storleken 28x28 pixlar. Pixlarna är i gråskala med värden från 0 till 255, som för bättre prestanda normaliserades till värden mellan 0 ch 1. Vi delade upp bilderna i 60 000 träningsbilder och 10 000 testbilder. Dessa laddades in i programmet genom

3.2 Datarensning

Därefter försöker man rensa data från information som inte är användbar, eller inte är komplett. I det andra fallet, och om man redan har tillräckligt många datapunkter, eliminerar man den datapunkt som inte är komplett. Alternativet är att man försöker med olika metoder ”gissa” eller resonera sig fram till vad de saknade värdena kan vara och fyller i det som saknas.

I vårt MNIST dataset fanns det av ganska naturliga skäl inga inga saknade datapunkter, dvs det fanns inga bilder där de hade värden utanför intervallet 0 och 255. Däremot fanns det en misstanke om att en eller flera rader och kolumner som ligger närmast kanterna på alla bilder kunde med fördel separeras för att minska antalet punkter som behövde behandlas under träningen. Detta visade sig inte stämma, eftersom redan vid en eliminering av tre pixlar från alla fyra kanter ledde till en kraftig försämring av noggrannheten.

3.3 Förbehandling

Till sist organiserar man data så att man kan förbereda den för att utvinna största möjliga information i de efterföljande stegen. När man väl har komplett och rensad data kan man fortsätta med att applicera olika metoder för att minska tid- och minneskostnaden vid

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Tabell 1: Kod för att ladda in data från MNIST-datasetet

träningstillfället, som brukar vara den mest resurskrävande delen vad gäller just tid och minne.

Eftersom vår data redan är i gråskala behövs ingen analys av färger. Däremot kan det vara värt att titta på hur mycket av ytan på varje bild som tas upp av siffrorna. För detta superpositionerade vi alla bilderna och tittade på .....??? alla bilder och medelvärdet av pixlarna beräknades. Det visade sig att siffrorna tar upp en relativt liten del av bilden, vilket kan vara en anledning till att modellerna inte presterade bättre än de gjorde ???

Det finns flera olika metoder utav vilka vi använde PCA för den smidighet men samtidigt kapacitet den presenterar.

??? Stratifide K-fold

??? optuna

??? streamlit

3.4 Experiment

Vi gjorde våra experiment med betydligt färre data, dvs 18 000 träningsbilder, som delades upp i tre lika stora delar. Detta för att spara tid vid många iterationer och test av olika modeller och metoder. På detta sättet tog varje modell högst 10 minuter per iteration.

3.5 Val av modell(er)

Nästa steg blir att använda en eller flera maskininlärningsmodeller för att memorera den underliggande strukturen i data och bygga en färdig motor som kan generalisera och, förhoppningsvis, med god sannolikhet klassificera eller förutse värden för bilder som matas in.

Här använde vi oss av de tre modeller som presenterades redan i teoridelen. Dessa ansåg vi vara de bästa och samtidigt snabbaste man kunde använda. Till sist

3.6 Steg 2

4 Resultat

5 Diskussion

We can use the statistics for each pixel to see if it varies that much over the dataset. And if they are all on a row or column, we can remove that row or column to reduce number of features.

Remember that since the model did perform better on test set compared to the training set, it is definitely not overfitted to the data and has been able to generalize

Tabell 2: Tider är räknade i sekunder och högsta och minsta värdet på varje rad är markerad med grön, respektive röd

			Normal	Trunkerad	Roterad	Trunkerad och roterad
SVM						
	Träning	Tid	132.4	155.9	180	155.2
		Noggrannhet	97.75%	97.74%	97.77%	97.77%
	Test	Tid	40.9	51.6%	55	52.7
		Noggrannhet	97.74%	97.70%	97.71%	97.72
RF						
	Träning	Tid	172.6	170.6	187.8	170.7
		Noggrannhet	93.33%	93.30%	93.06%	93.18%
	Test	Tid	44.8	43.8	48.9	43.6
		Noggrannhet	93.59%	93.74%	93.60%	93.51%
XGBoost						
	Träning	Tid	717.2	263.4	347.5	273.3
		Noggrannhet	96.44%	96.38%	96.46%	96.38%
	Test	Tid	56.2	31.4	35.1	33.2
		Noggrannhet	94.70%	94.95%	94.66%	94.94%
Ensemble						
	Test	Tid	292.3	189.3	326.3	189.1
		Noggrannhet	97.29%	97.33%	97.34%	97.29%

Tabell 3: Normal

	precision	recall	f1-score	support
0	0.98	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.96	0.98	0.97	1010
4	0.98	0.97	0.97	982
5	0.98	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.97	0.96	0.97	1028
8	0.96	0.97	0.97	974
9	0.96	0.96	0.96	1009

Use feature selector
change the heatmap so that it shows only bottom half of the matrix, using

```
# Create the correlation matrix
corr = ansur_df.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Add the mask to the heatmap
sns.heatmap(corr, mask=mask, cmap=cmap, center=0, linewidths=1, annot=Tru
plt.show()
```

The code seems redundant, but it's been done because different model has be applied on different data, different sizes, and in different steps. It assured that the procedures would not mix up. But in a future implementation these redundancies can be removed and replaced with two or three functions that based on the name of the model could perform the same procedure.

Tabell 4: Trunkerad

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.97	0.98	0.97	1010
4	0.98	0.97	0.97	982
5	0.98	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.97	0.96	0.97	1028
8	0.97	0.97	0.97	974
9	0.96	0.96	0.96	1009

Tabell 5: Roterad

	precision	recall	f1-score	support
0	0.98	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.96	0.98	0.97	1010
4	0.98	0.97	0.97	982
5	0.98	0.97	0.98	892
6	0.98	0.98	0.98	958
7	0.97	0.96	0.97	1028
8	0.96	0.97	0.97	974
9	0.96	0.96	0.96	1009

In datacamp he says the random forest is highly accurate already with the default scikit-learn parameters

The time and memory complexity of machine learning algorithms can vary based on several factors, including the specific implementation, the size and complexity of the dataset, and the hyperparameters used. Here's a general overview of how the time and memory complexity might increase with an increased number of observations for Random Forest, Support Vector Classifier (SVC), and XGBoost:

Random Forest:

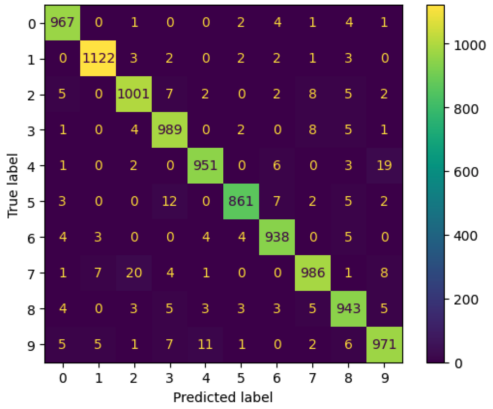
Time Complexity: The time complexity of building a Random Forest model generally increases linearly with the number of observations (n) and features (p). For each decision tree in the forest, the time complexity of training a single tree is typically O(n * p * log(n)). Memory Complexity: Random Forests can require a significant amount of memory, especially when dealing with large datasets or a large number of trees in the forest. The memory complexity is typically O(n * p * m), where m is the number of trees in the forest. Support Vector Classifier (SVC):

Time Complexity: The time complexity of training an SVC can increase quadratically or worse with the number of observations (n) and features (p) in the worst case, especially for non-linear kernels like the radial basis function (RBF) kernel. The time complexity is typically $O(n^2 * p)$ or worse. Memory Complexity: The memory complexity of an SVC depends on the kernel used and the size of the dataset. For large datasets, especially when using non-linear kernels, the memory complexity can be significant. XGBoost:

Time Complexity: XGBoost is an efficient gradient boosting algorithm that can handle large datasets. The time complexity of training an XGBoost model depends on the number of observations (n) and features (p), as well as the number of boosting iterations (num_boost_round). It is typically linear or slightly worse than linear in the number of

Tabell 6: Trunkerad och roterad

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.97	0.98	0.97	1010
4	0.98	0.97	0.97	982
5	0.98	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.97	0.96	0.97	1028
8	0.97	0.97	0.97	974
9	0.97	0.96	0.96	1009



(a) and (b)

Figur 3: Normal

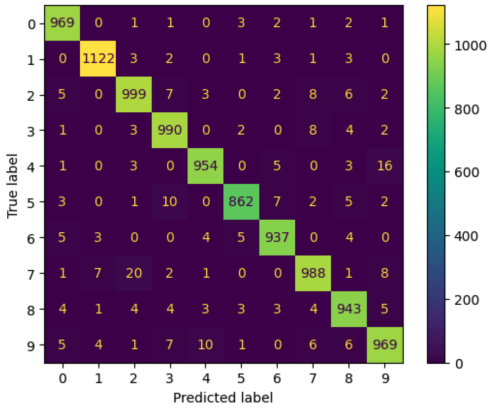
boosting iterations. Memory Complexity: XGBoost can require a significant amount of memory, especially when dealing with large datasets or a large number of features. The memory complexity is typically O(n * p) for storing the dataset and additional memory for storing intermediate results during training. Overall, Random Forest tends to have lower time complexity compared to SVC for large datasets, while XGBoost can be more memory-efficient and faster to train compared to both Random Forest and SVC in many cases, especially for large datasets. However, the actual performance may vary depending on the specific characteristics of the dataset and the hyperparameters used for each algorithm.

Explained variance ratio (Izenman, 2008; Kuhn and Johnson, 2019; OpenClassrooms, 2021)

The explained variance ratio (EVR) in the context of PCA (Principal Component Analysis) is a measure that indicates the proportion of the dataset's variance that is captured by each principal component. Specifically, for each principal component, the explained variance ratio represents the ratio of the variance along that component to the total variance in the dataset.

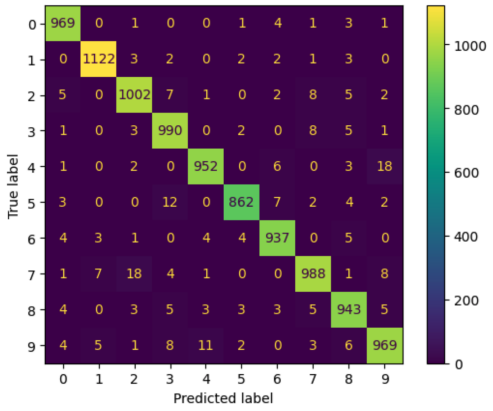
Mathematically, if λ_i represents the eigenvalue associated with the i-th principal component, and $\lambda_1, \lambda_2, ..., \lambda_n$ are the eigenvalues sorted in descending order, then the explained variance ratio for the i-th principal component is given by $EVR_i = \frac{\lambda_i}{\Sigma_{j=1}^{n} \lambda_j}$

In simpler terms, the explained variance ratio indicates how much information (variance) is retained when the dataset is projected onto a lower-dimensional subspace defined by the principal components. It helps us understand the relative importance of each principal component in capturing the overall variance in the dataset.



(a) and (b)

Figur 4: Trunkerad



(a) and (b)

Figur 5: Roterad

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(std_df)
print(pca.explained_variance_ratio_)
```

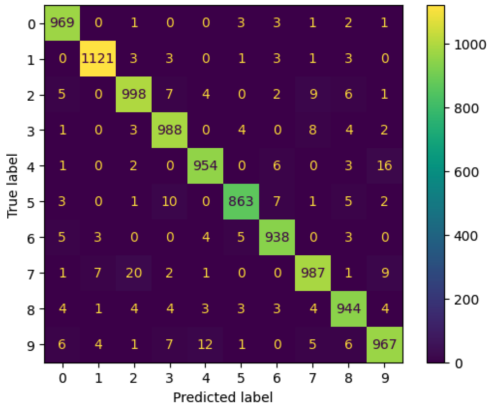
??? Please observe that divding by 255 is not scaling but standardaizing the data. Scaling is done by subtracting the mean and dividing by the standard deviation. This is done to make the data have a mean of 0 and a standard deviation of 1. This is important for some algorithms, like SVM, that are sensitive to the scale of the input features. In this case, the data is already in the range of 0 to 1, so scaling is not necessary.

6 Slutsats

6.1 Framtida arbete

I början var koden olika för olika modeller. Men det tog ett bra tag innan koderna blev enhetliga, vilket med tanke på tidsbrist inte kunde organiseras på rätt sätt. Det goda resultatet har inte påverkats men det kunde organiseras bättre i form av funktioner som kunde förenkla de omskrivningar som har gjorts för varje modeller. Detta är en del av det pågående och framtida arbetet.

Would be interesting to use EVR for each number separately and see if an ensemble learning with 10 members—for the 10 different—digits could learn the patterns faster



(a) and (b)

Figur 6: Trunkerad och roterad

and produce a result that is more reliable and accurate than the one we have used here.

Referenser

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Crammer, K. and Singer, Y. (2003). A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley.

Gibbs, A. and MacKay, D. J. (1997). Variational gaussian process classifiers. *IEEE Transactions on Neural Networks*.

Guido, S. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

IBM (2021). Random forest. *IBM*.

Izenman, A. J. (2008). *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer.

Kuhn, M. and Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press.

OpenClassrooms (2021). Analyze the results. *OpenClassrooms*.