COMP5318 Assignment 1

lihang shen

September 2024

1 Report

In assignment 1, there is a training data set whose name is ”train.csv” and a testing data set whose name is ”test1.csv”. In ”train.csv”, there are 30,000 samples with 784 attributes each. Every attribute represents a pixel in a 28*28 image and the value for each attribute is between 0 and 255. It also includes a label for each sample which name is ”label” in the data set. The value for ”label” is an integer between 0 and 9. 0 represents a t-shirt, 1 represents a trouser, 2 represents a pullover, 3 represents a dress, 4 represents a coat, 5 represents a sandal, 6 represents a shirt, 7 represents a sneaker, 8 represents a bag, 9 represents an ankle boot. Using integer labels for the image classes helps to save memory and improves computational efficiency, especially when handling large datasets. By the way, ”test1.csv” is similar to ”train.csv” except it only has 2,000 samples. In this assignment, I implemented 4 machine-learning methods in total. They are Nearest Neighbour (knn), Logistic Regression, SVM, and Boosting. Among these methods, SVM performed the best, achieving approximately 90% accuracy on the test set. The other 3 methods get about 85% accuracy on the test set. This suggests that SVM is especially well-suited for 28*28 image classification, and other methods have slightly poor performance for image classification.

Before I build classifiers, I first use some data cleaning and preprocessing methods. They are removing NAs, normalization, and PCA. As for removing NA, I do this because any sample that include NA may lead to the failure of building the classifier. For example, in the knn method, we need to calculate the distance between the new sample and the training samples, but if the training samples missing some attribute values, we can not do that calculation. In addition, I used the normalization method. The equation for normalization is:

- x means an attribute of a sample
- X means a collection of all samples' x attributes

$$x_{norm} = \frac{x - min(X)}{max(X) - min(X)} \tag{1}$$

It illustrates 0 will represent the lowest value in the original data set because $x - min(X) = 0$ and 1 will represent the largest value in the original data set because $x - min(X) = max(X) - min(X)$. According to the Week 2 lecture, ”Normalization transforms original data to a new range and it is used to avoid the dominance of attributes with large values over attributes with small values”. As a result of using this method, I need not worry about the influence of extreme value (i.e. outliers) being too big. By the way, normalization can also make attributes have the same scale and improve the gradient descent speed. Furthermore, I used PCA methods for the normalized data set. According to the Week 6 lecture, ”PCA projects the data into a lower dimensional space that still captures the important information”. After using this method, I successfully reduce the number of attributes from 784 to 187. The principle of this method is very complex, we need to first calculate all principle components from attributes by using:

- n: number of samples
- m: number of attributes
- $X_{n\times m}$: data matrix
- $U_{n\times n}$: the transformed data, i.e. the i-th row of U contains the coordinates of the i-th row of X in the new coordinate system
- $\wedge_{n\times m}$: diagonal matrix containing the singular values (either positive number or zero number)
- $V_{m\times m}$: defines the new set of axes (principal components)

$$X_{n\times k} = U_{n\times k} \times \wedge_{k\times k} \times V^T_{k\times k} \tag{2}$$

Then I need to use some methods, such as the Elbow method, to choose the principle components by comparing their variances. To get most principle components, I choose to preserve 95% of variances. On one side, it greatly preserves the most important principle components, on the other side, it greatly reduces the number of attributes. It also helps classifiers to avoid overfitting by reducing the number of unnecessary components, it works because classifiers are less likely to fit unnecessary components (i.e. noise). Lastly, I did not do standardization before using PCA because every attribute has the same scale which means we need not standardize them except doing those data preprocessing methods. I also chose to split the training data set to a sub-training data set, which occupies 90% in original training data set, and a validation data set, which occupies 10% in original training data set. To avoid THE skew problem, I choose to use the "train_test_split" method from thr sklearn library. This could help us know how well each model will perform on real-world data.

Next, I want to explain about the hyper-parameters tuning method. In this assignment, I mainly use the combination of grid-search and cross-validation. Gird-search is pretty simple, I just need to set a dictionary where the keys are hyper-parameters, and the values are lists of possible values for those hyper-parameters. Grid-search method tries to find all possible combinations of hyper-parameters provided in that dictionary with different values. Then I need to use cross-validation to get an overall score for each combination. Specifically, I will do:

- 1. Split data into k sets set1,.., setk of approximately equal size
- 2. A classifier is built k times. Each time the testing is on 1 set and the training is on the remaining k sets together
- 3. calculate accuracy acc_i for each classifier and finally calculate the overall performance score $Avg(acc1, \ldots, acci, \ldots, acck)$ for current combination of hyper-parameters.

After doing those steps for all possible combinations of hyper-parameters, the one with the best overall score is the best classifier among those choices.

Next, I want to talk about model selections, I first choose knn because it is good at classifying images. According to LinkedIn (n.d.), "KNN can be very effective for small and simple datasets, and for images that have clear and distinct features". For example, an unlabeled picture (like a T-shirt) will only be identified as a T-shirt when most of its pixel has the same gray scale as some labeled T-shirt pictures in the training data set. It works because similar pixel values will lead to a small distance between an unlabeled image and its true class. The theory for this method is pretty simple, it just calculates the distance between unlabeled image B and labeled image A in the training data set by using the formula $D_E(A, B) = \sqrt{(a_1 - b_1)^2 + ... + (a_n - b_n)^2}$. After comparing image B with all images in the training data set, we get k neighbors with the smallest D_E. Finally, the classifier will use the majority class in the k nearest neighbors to predict the class of unlabeled image. Lastly, in the code part, I will use the grid search method to find the best combination among "n_neighbors", "weights", and "metric". Among those hyper-parameters, "n_neighbors" simply means k; "weights" means the method for doing classify, such as "uniform" means each neighbor has equal weight and "distance" means closer neighbor has higher weight; "metric" simply means the method to calculate the distance (in the above description, I mainly use Euclidean as an example)

Secondly, I choose SVM because it can handle data with many features. According to the Week 6 lecture slide, "SVM transform the data from its original feature space to a new space where a linear boundary can be used to separate the data". Namely, using this methods may help us identify the differences between each class. In plain English, SVM is to find the maximum margin hyperplane between Support vectors. In mathematical description, we use $H = w \cdot x + b = 0$ to represent the decision boundary, and we need to maximize $d = \frac{2}{||w||^2}$ with restriction $y_i(w \cdot x_i + b) \geq 1$, where d means the margin and w is the coefficient of boundary function. By the way, the restriction simply means we need to correctly clarify every sample in the training set. After using Lagrange multipliers to transfer it to an unrestricted equation, we can get the classifier equation $f = w \cdot z + b = \sum_{i=1}^{N} \lambda_i y_i x_i \cdot z + b$. Since we cannot separate our data set in a linear space. Therefore, I choose to use the kernel trick to calculate the classifier when I project the original space to a higher-dimensional space. The new classifier equation is $f = w \cdot z + b = \sum_{i=1}^{N} \lambda_i y_i K(x_i, z) + b$, where K means a kernel function. In addition, I use "grid search" (only changing hyper-parameter kernel function due to PC capability) to judge which kernel function I should use to get a better classifier. Finally, I use soft margin methods to avoid overfitting, this method just makes the decision boundary not strict. Specifically, we are allowed to have some misclassified when we train the decision boundary equation.

Thirdly, I choose logistic regression because it is very simple and easy to train which suits to personal computer. According to GeeksforGeeks (2024), "Logistic regression is easier to implement, interpret, and very efficient to train." The theory of this method is very simple. Firstly, set an initial weight W which represents the coefficients for the classifier. Secondly, minimize cross-entropy loss function $f(w) = -\frac{1}{n} \sum_{i=1}^{n} *[y_i log(W^T X^{(i)} + b) + (1 - y_i) log[1(W^T X_i + b)]]$ which means to calculate $\nabla f(w) = \sum_{i=1}^{n} (\sigma(W_t^T X^{(i)}) - y^{(i)}) X^{(i)}$. Thirdly, we need to get a new W by using formula $W_{t+1} = W_t - \alpha_t \nabla f(w) = W_t - \alpha_t \sum_{i=1}^{n} (\sigma(W_t^T X^{(i)}) - y^{(i)}) X^{(i)}$, where α is the study rate. We keep repeating the above steps k times to get the best coefficient vector w. As for α we need not care about it in the code part, because there is no hyper-parameter related to it. However, for k, I will use different loop times to get the best classifiers. Lastly, I will change the hyper-parameter C to avoid overfitting (a larger C corresponds to less regularization).

Lastly, I choose boosting methods. It is a type of ensemble method that mainly focuses on manipulating the training data. The weak classifier I choose for the weak learner is a 1-depth decision tree, we call it a weak learner because it usually has bad performance. The reason I chose this method is boosting method is one of the most widely used methods. It has a strong performance in classification. Next, I will explain how Ada boosting works in detail. Firstly, we usually begin by initializing weight for each sample and it is usually $\frac{1}{N}$. Then we begin to do iteration, details about iteration are represented below:

- T: the max iteration times,
- t: the number of iterations, from 1 to T,
- $y_n \in \{-1, 1\}$: y_n means the true target label of x_n,

- x_n: the data point,
- $w_t(n)$: the weight in the t^{th} iteration,
- $h_t(x)$: the weak classifier in the t^{th} iteration,
- $I[y_n \neq h_t(x_n)]$: it only has 2 results. -1 means predicted label equals true label of sample x_n; 1 otherwise.
- ϵ_t: weighted classification error, measures the total misclassification error

```
1: for For t = 1; t ≤ T; t++ do
2:     Training weak classifier $h_t(x)$ by using $w_t(n)$ and training data.
3:     Using $I[y_n \neq h_t(x_n)]$ to calculate weighted classification error $\epsilon_t = \sum w_t(n) I[y_n \neq h_t(x_n)]$
4:     Then calculate the contribution for this weak classifier $\beta_t = \frac{1}{2} log \frac{1-\epsilon_t}{\epsilon_t}$
5:     Update weights for next loop, that is normalize $w_{t+1}(n) = w_t(n) e^{-\beta_t y_n h_t(x_n)}$ such that $\sum_n w_{t+1}(n) = 1$
6: end for
7: Compute the strong classifier: $h[x] = sign[\sum_{t=1}^{T} \beta_t h_t(x)]$
```

As you can see, we finally combine those weak classifiers together to get a strong classifier. However, after I used this method in coding part, I found the accuracy of that strong classifier is only about 50%. As a result, I chose to use another boosting method - the gradient boosting method which usually has better performance. The difference between ada boosting and gradient boosting is mainly reflected in the method to deal with errors in each iteration (Data HeadHunters, 2024). As for the gradient boosting method, it mainly focuses on minimizing loss function. That is to say, using prediction error to generate a new regression problem. For this method, I did not use grid search because my PC cannot calculate them efficiently.

Next, I will explain the experimental settings in terms of the implementation strategy and the hyper-parameter fine-tuning strategy. Firstly, I start by using the sklearn library to build an initial version of each model. Secondly, I will gather information from both sklearn official documents and other related websites to decide which hyper-parameters require tuning. Thirdly, I will decide how many values to test for each hyper-parameter based on the performance of the initial model. If it performs well, then I will explore a wider range of values. I do this because I think this model has more potential compared to those with poorer performance. Fourthly, I will decide k for cross-validation based on the training time of the initial model. If the training time is slow, then I will use a small k because my computer cannot handle extensive training efficiently. Fifthly, I will check the result of grid-search and cross-validation by setting "verbose = 3". If the model gets improved, then I will try to find the trend of the best combination of hyper-parameters. For example, during analyzing the SVM grid-search result, I find the classifier with a larger "C" likely has better accuracy, hence I guess "C" is very important for the SVM method; therefore, I decided to manually build classifiers with different values of "C". If I get a better classifier during the construct process, I will include these values in the grid search dictionary and perform the grid search again. This process continues until no further improvements in hyper-parameter combinations are found.

After briefly introducing the methodologies and experimental settings, I prefer to illustrate my result for this assignment. Firstly, let us see a few graphs and matrices.

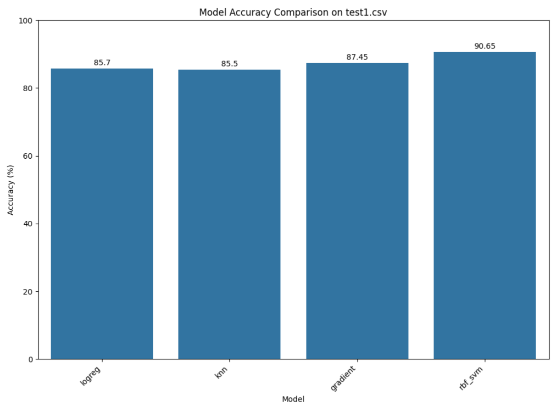

Figure 1: test1.csv prediction accuracy

Figure 1 is the prediction accuracy result of each classifier on test1.csv. Among those classifiers, rbf_svm has the best performance with an accuracy of 90.65%. knn has the worst performance with an accuracy of 85.5%.

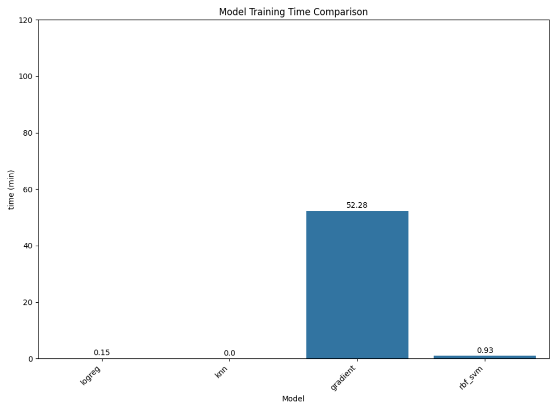

Figure 2: model training time

Figure 2 represents the training time for each model, it does not include the inference time because inference time is highly related to the training time. Specifically, when I infer the best model, I just simply train different models with certain combinations of hyper-parameters. From this figure, we can easily find that the gradient boosting method takes a long time to train. The reason is that convergence is slow with gradient descent, especially when we set a very small learning rate. Knn has the fastest training time which is less than 1 minute. The reason behind this is that calculating distance is simple.

$$\begin{bmatrix} 168 & 1 & 2 & 6 & 0 & 0 & 13 & 0 & 1 & 0 \\ 1 & 192 & 2 & 3 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 149 & 2 & 21 & 0 & 16 & 1 & 0 & 0 \\ 8 & 3 & 2 & 151 & 9 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 15 & 8 & 165 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 174 & 0 & 19 & 1 & 15 \\ 33 & 0 & 35 & 2 & 15 & 0 & 115 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 201 & 0 & 9 \\ 0 & 0 & 3 & 1 & 0 & 0 & 5 & 1 & 206 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 13 & 0 & 189 \end{bmatrix}$$

Confusion Matrix for Knn in test1.csv

$$\begin{bmatrix} 162 & 2 & 0 & 10 & 0 & 1 & 13 & 0 & 3 & 0 \\ 0 & 194 & 1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 147 & 4 & 20 & 0 & 14 & 0 & 2 & 0 \\ 4 & 5 & 2 & 150 & 6 & 0 & 7 & 0 & 0 & 0 \\ 1 & 0 & 16 & 8 & 162 & 0 & 15 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 183 & 0 & 16 & 1 & 8 \\ 26 & 1 & 28 & 2 & 11 & 1 & 129 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 192 & 0 & 10 \\ 0 & 0 & 2 & 3 & 0 & 0 & 4 & 0 & 206 & 1 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 11 & 0 & 189 \end{bmatrix}$$

Confusion Matrix for logistic regression in test1.csv

$$\begin{bmatrix} 171 & 2 & 1 & 5 & 0 & 1 & 10 & 0 & 1 & 0 \\ 0 & 195 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 161 & 3 & 11 & 0 & 12 & 0 & 1 & 0 \\ 3 & 4 & 3 & 157 & 6 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 8 & 4 & 181 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 201 & 0 & 5 & 1 & 3 \\ 25 & 0 & 19 & 3 & 13 & 0 & 139 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 202 & 0 & 8 \\ 0 & 0 & 1 & 1 & 0 & 0 & 4 & 0 & 209 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 197 \end{bmatrix}$$

Confusion Matrix for svm in test1.csv

$$\begin{bmatrix} 167 & 1 & 0 & 9 & 0 & 1 & 12 & 0 & 1 & 0 \\ 0 & 192 & 2 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 151 & 2 & 22 & 0 & 11 & 0 & 2 & 0 \\ 4 & 4 & 3 & 150 & 7 & 0 & 6 & 0 & 0 & 0 \\ 1 & 0 & 12 & 7 & 162 & 0 & 21 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 194 & 0 & 7 & 0 & 6 \\ 21 & 1 & 20 & 1 & 14 & 0 & 142 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 197 & 1 & 11 \\ 1 & 0 & 4 & 1 & 0 & 3 & 5 & 1 & 201 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 7 & 0 & 193 \end{bmatrix}$$

Confusion Matrix for gradient boosting in test1.csv

From those matrices, you can see that all classifiers cannot precisely detect class 6 - shirt and always predict class 6 as class 0 (t-shirt), 2 (pullover), or 4 (coat). I think it is reasonable because even as a human, I cannot 100% detect the differences between some shirts and t-shirts.

In conclusion, there are 2 findings. Firstly, all classifiers get more than 85% accuracy on the test1.csv. Secondly, all classifiers cannot precisely classify class 6. After finishing the assignment, I decided to choose SVM as the best classifier. There are 2 main reasons. Firstly, its training time is short. Secondly, its accuracy on the testing data set is pretty good. In addition, I also want to describe the result of my best classifier's performance in Kaggle. Until I finished this report, I rank at 50th place in the leader board with 97.15% accuracy. I think that's a pretty good result. Furthermore, there are 3 limitations to this assignment. The first is the size of the training dataset. With a larger dataset, I believe the classifier's performance would improve. The second limitation concerns the quality of the samples. Upon manually inspecting some of the training images, I found that some have poor quality. This could adversely affect the training results to some extent. Lastly, on the training data set, I think there are some samples with incorrect label. In Code Part 3.1, the example provided is a pullover (class 2), but it is labeled as class 6 in the picture. Correcting these incorrect labels is nearly impossible; therefore, I consider this a significant limitation. Regarding the methods and results, based on the outcomes, most methods produced effective classifiers. I selected SVM as the best performer due to its relatively short training time and higher accuracy. Lastly, for future improvements, one approach could be to identify a classifier that excels specifically in predicting class 6. This specialized classifier could then be combined with the best-performing classifier from my current models. In the combined model, the predictions made by the specialized classifier for class 6 would carry more weight compared to the original classifier. I believe this method could enhance overall performance by improving predictions for class 6.

2 Appendix

Instructions: please use following python version and download all required packages with correct version. You can use "pip install packagename" to download those packages. Training time may change because of different hardware.

Hardware Infomation: Processor is "11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz 3.60 GHz"; Installed RAM is "32.0 GB (31.9 GB usable)"; "System type" is 64-bit operating system, x64-based processor

Python version: Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32

External package: Below list include all packages and libraries in my pc related to this assignment, you should use pip to install them:

Package	**Version**
pandas	2.2.2
numpy	2.0.1
matplotlib	3.9.2
scipy	1.14.0
scikit-learn	1.5.1
seaborn	0.13.2

However, if you still cannot run my assignment, please download all packages below

Package	**Version**
annotated-types	0.5.0
anyio	4.4.0
argon2-cffi	23.1.0
argon2-cffi-bindings	21.2.0
arrow	1.3.0
asttokens	2.4.1
async-lru	2.0.4
attrs	23.1.0
Babel	2.15.0
beautifulsoup4	4.12.3
black	23.9.1
bleach	6.1.0
certifi	2023.7.22
cffi	1.16.0
charset-normalizer	3.2.0
click	8.1.7
colorama	0.4.6
comm	0.2.2
contourpy	1.2.1
cowsay	6.0
cycler	0.12.1
debugpy	1.8.2
decorator	5.1.1
defusedxml	0.7.1
emoji	2.8.0
executing	2.0.1
fastjsonschema	2.20.0
fonttools	4.43.1
fpdf	1.7.2
fpdf2	2.7.6
fqdn	1.5.1
h11	0.14.0
httpcore	1.0.5
httpx	0.27.0
idna	3.4
inflect	7.0.0
iniconfig	2.0.0
ipykernel	6.29.5
ipython	8.26.0
isoduration	20.11.0
jedi	0.19.1
Jinja2	3.1.4
joblib	1.4.2
json5	0.9.25
jsonpointer	3.0.0
jsonschema	4.19.1
jsonschema-specifications	2023.7.1
jupyter$_{client}$	8.6.2

Package	**Version**
jupyter$_core$	5.7.2
jupyter-events	0.10.0
jupyter-lsp	2.2.5
jupyter$_server$	2.14.2
jupyter$_server_terminals$	0.5.3
jupyterlab	4.2.4
jupyterlab$_pygments$	0.3.0
jupyterlab$_server$	2.27.3
kiwisolver	1.4.5
MarkupSafe	2.1.5
matplotlib	3.9.2
matplotlib-inline	0.1.7
mistune	3.0.2
mypy	1.6.0
mypy-extensions	1.0.0
nbclient	0.10.0
nbconvert	7.16.4
nbformat	5.10.4
nest-asyncio	1.6.0
notebook	7.2.1
notebook$_shim$	0.2.4
numpy	2.0.1
overrides	7.7.0
packaging	23.1
pandas	2.2.2
pandocfilters	1.5.1
parso	0.8.4
pathspec	0.11.2
Pillow	10.0.1
pip	23.2.1
platformdirs	3.10.0
pluggy	1.3.0
prometheus$_client$	0.20.0
prompt$_toolkit$	3.0.47
psutil	6.0.0
pure$_eval$	0.2.3
pycparser	2.22
pydantic	2.3.0
pydantic$_core$	2.6.3
pyfiglet	1.0.2
Pygments	2.18.0
pyparsing	3.1.2
pytest	7.4.2
python-dateutil	2.9.0.post0
python-json-logger	2.0.7
pytz	2024.1
pywin32	306
pywinpty	2.0.13

Package	**Version**
PyYAML	6.0.1
pyzmq	26.0.3
referencing	0.30.2
requests	2.31.0
rfc3339-validator	0.1.4
rfc3986-validator	0.1.1
rpds-py	0.10.4
scikit-learn	1.5.1
scipy	1.14.0
seaborn	0.13.2
Send2Trash	1.8.3
setuptools	65.5.0
six	1.16.0
sniffio	1.3.1
soupsieve	2.5
stack-data	0.6.3
tabulate	0.9.0
terminado	0.18.1
threadpoolctl	3.5.0
tinycss2	1.3.0
tornado	6.4.1
traitlets	5.14.3
types-python-dateutil	2.9.0.20240316
typing$_{e}xtensions$	4.8.0
tzdata	2024.1
uri-template	1.3.0
urllib3	2.0.4
validator-collection	1.5.0
wcwidth	0.2.13
webcolors	24.6.0
webencodings	0.5.1
websocket-client	1.8.0

3 Reference

LinkedIn. (n.d.). You need to identify images quickly, which machine hqtie. Retrieved September 15, 2024, from https://www.linkedin.com/advice/1/you-need-identify-images-quickly-which-machine-hqtie

GeeksforGeeks. (2024, August 30). Advantages and disadvantages of logistic regression. Retrieved September 15, 2024, from https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/

Data HeadHunters. (2024, January 4). Gradient boosting vs. AdaBoost: Battle of the algorithms. Retrieved September 15, 2024, from https://dataheadhunters.com/academy/gradient-boosting-vs-adaboost-battle-of-the-algorithms/