Computer Project Report

Hieu Le Trung, Tuyen Dao Cong

June 2021

Abstract

This report is the full result we have completed during the MAI391 (Mathematics for Machine Learning) course. Project contains a number of foundational algorithms in Machine Learning including platform theory while also expressing the implementation of various versions of the algorithm. In the project is the final result and the best result during the implementation process.

Contents

1 Linear regression model with real datasets

1.1 Introduction

1.1.1 Linear Regression

Regression modeling deals with the description of the sampling distribution of a given random variable y and how it varies as function of another variable or a set of such variables $\mathbf{x} = \begin{bmatrix} x_0 & x_1 & ... & x_{n-1} \end{bmatrix}^T$. The first variable is called the **dependent**, the **outcome** or the **response** variable while the set of variables $\mathbf{x}$ is called the **independent** variable, or the **predictor** variable or the **explanatory** variable. [2]

A regression model aims at finding a likelihood function $\mathbf{p(y|x)}$, that is the conditional distribution for $\mathbf{y}$ with a given $\mathbf{x}$. The estimation of $\mathbf{p(y|x)}$ is made using a data set with:

- N cases i = 0,1,2,. . . , N - 1
- Response (target or outcome) variable y_i with i = 0,1,2,. . . ,N - 1
- p so-called explanatory (predictor) variables, $\mathbf{x_i} = \begin{bmatrix} x_i, 0 & x_i, 1 & ... & x_{i,p-1} \end{bmatrix}$ with i = 0,1,2,. . . , N - 1 and explanatory variables running from 0 to $p-1$.

Consider an experiment in which p characteristics of n samples are measured. The data from this experiment, for various explanatory variables p are normally represented by a matrix $\mathbf{X}$.Each data point is a row of X, we have $\mathbf{X} = \begin{bmatrix} \mathbf{x_0}; & \mathbf{x_1}; & ... & ; \mathbf{x_{N-1}} \end{bmatrix}$ (The matrix $\mathbf{X}$ is called the *design matrix*) and the result $\mathbf{y_i} = y(\mathbf{x_i})$ with i = 0,1,2,. . . , N 1. Additional information of the samples is available in the form of $\mathbf{y}$. The aim of regression analysis is to explain $\mathbf{y}$ in terms of $\mathbf{X}$ through a functional relationship like $\mathbf{y} = f(X_{i,:})$. When no prior knowledge on the form of f() is available, it is common to assume a linear relationship between $\mathbf{X}$ and $\mathbf{y}$. This assumption gives rise to the linear regression model where $\mathbf{w} = \begin{bmatrix} w_0 & w_1 & ... & w_{p-1} \end{bmatrix}$ are the regression parameters. Linear regression gives us a set of estimations for the parameters w_j for $j = 0, 1, ..., p-1$. The approach to estimate is to assume the result can be estimated from each input data point by a linear function, that is:

$$y(x_i) \approx \hat{y} + e_i = \overline{\mathbf{x_i}}\mathbf{w} + e_i \tag{1}$$

where e_i is the error in our approximation. Defining the vectors:

$$\mathbf{y} = \begin{bmatrix} y_0 & y_1 & ... & y_{N-1} \end{bmatrix} \tag{2}$$

$$\mathbf{w} = \begin{bmatrix} w_0 & w_1 & ... & w_{p-1} \end{bmatrix} \tag{3}$$

$$\mathbf{e} = \begin{bmatrix} e_0 & e_1 & ... & e_{N-1} \end{bmatrix} \tag{4}$$

and the design matrix

$$\mathbf{X} = \begin{bmatrix} 1 & x_{0,1} & ... & x_{0,p-1} \\ 1 & x_{1,1} & ... & x_{1,p-1} \\ ... & ... & ... & ... \\ 1 & x_{N-1,1} & ... & x_{N-1,p-1} \end{bmatrix} \tag{5}$$

we rewrite again our equations as

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{e} \tag{6}$$

The left-hand side of this equation is kwown. Our error vector $\mathbf{e}$ and the parameter vector $\mathbf{w}$ are our unknown quantities. The question is that how can we find the optimal set of w_i value to minimize the error vector $\mathbf{e}$. The loss function is:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=0}^{N-1}(\overline{\mathbf{x_i}}\mathbf{w} - \mathbf{y})^2 = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \frac{1}{2}\|e\|_2^2 \tag{7}$$

The derivative of loss function:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \tag{8}$$

and for the loss function to be extreme, we have to find the solution of the derivative equation equal to 0. That's equivalent to:

$$\mathcal{L}'(\mathbf{w}) = 0 \Longleftrightarrow \mathbf{X^T}\mathbf{X}\mathbf{w} = \mathbf{X^T}\mathbf{y} \tag{9}$$

for $\mathbf{X}, \mathbf{y}$ is known. [3]

If $\mathbf{X^T}\mathbf{X}$ is *invertible*, then we have the unique linear regression's solution $\mathbf{w} = (\mathbf{X^T}\mathbf{X})^{-1}\mathbf{X^T}\mathbf{y}$. However, if $\mathbf{X^T}\mathbf{X}$ is not invertible, it means its determinant is 0, we use the *pseudo-inverse concept* of a matrix. This is related to the concept *Singular Value Decomposition* of a matrix. Note that we can show any matrix can always be used to compute the SVD . A theorem has shown that a symmetric matrix is always can be *diagonalized*. On the other hand, we can also prove that the eigenvectors corresponding to different eigenvalues of a symmetric matrix are orthogonal to each other. For eigenvectors of a symmetric matrix correspond with an eigenvalue with its *algebraic multiplicity* equal to its *geometric multiplicity* (because the symmetric matrix is diagonalizable, for any eigenvalue, algebraic multiplicity is equal to its geometric multiplicity) and greater than 1 ,those vectors are a ?basis of *a solution space of a homogeneous system of linear equations*. Then, to be able to compute the SVD of the above matrix, we use the *Gram-Smith algorithm* to create a set of orthogonal vectors that lie in the same solution space. By this way, we can always compute the SVD of a matrix $\mathbf{A}$ by using its $\mathbf{A^T}\mathbf{A}$ and $\mathbf{A}\mathbf{A^T}$ in which they are all symmetric matrices [4, 1]. And it turns out that the minimum norm least squares solution of linear regression problem can be found in terms of the pseudo-inverse $\mathbf{A}^\dagger$ of $\mathbf{A}$, which is itself obtained from the SVD of A. Specifically, if

$$\mathbf{X^T}\mathbf{X} = \mathbf{M}\mathbf{K}\mathbf{N^T} \tag{10}$$

with $\mathbf{K} = diag(\lambda_1, \lambda_2, ..., \lambda_r, 0, 0, ..., 0)$ with $\lambda_i > 0, i = 1, 2, ..., r$ and $\mathbf{K}$ have the same size with $\mathbf{X^T}\mathbf{X}$. Then we have

$$\mathbf{K}^\dagger = diag(\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, ..., \frac{1}{\lambda_r}, 0, 0, ..., 0) \tag{11}$$

Finally, the *pseudo-inverse* of $\mathbf{X^T X}$ is defined as:

$$(\mathbf{X^T X})^\dagger = \mathbf{N K^\dagger M^T} \tag{12}$$

is the key to solve the solution of the problem. So all in all, that showed how linear regression can be done with a given data set.

1.1.2 Improved Linear Regression

The first limitation of Linear Regression is that it is very sensitive to outliers. Therefore, before performing Linear Regression, the outliers need to be removed. This step is called *pre-processing*. The second limitation of Linear Regression is that it cannot represent complex models. Although this approach can be applied if the relationship between outcome and input is not necessarily linear by calculate non-linear quantities in terms of linear quantities and then using it as an independent component. However, the relationship is still much simpler than in real models. Furthermore, the question is: how to define the dependent function correctly in the form of variable's components?

a/ Pre-Processing

Normally the data may need a rescaling and/or may be sensitive to extreme values. Scaling the data renders our inputs much more suitable for the algorithms to employ and the metrics to evaluate. **Scikit-Learn** has several functions which allow us to rescale the data, normally resulting in much better results in terms of various accuracy scores. The **StandardScaler** function in Scikit-Learn ensures that for each feature we using has the mean value is zero and the variance is one. This scaling has the drawback that it does not ensure that the data have a particular maximum or minimum in our data set. Another function included in Scikit-Learn is the **MinMaxScaler** which ensures that all features are exactly between 0 and 1

b/ Outlier Detection

There are many methods to identify and remove outliers. The simplest method is to rely on the *boxplot chart* to determine as well as remove outliers. Values less than 1.5 **interquartile range** from q_1 or less and values beyond the 1.5 **interquartile range** from q_3 are identified as outliers. The entire data point that has any outliers in its attributes are removed from the dataset.

1.1.3 Ridge Regression

Remind about the expression for the standard **Mean Squared Error (MSE)** (which is calculated by use our cost function divided by n) and the equations for the **ordinary least squares (OLS)** method:

$$\mathcal{C}(\mathbf{w}) = \frac{1}{n}((\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})) = \frac{1}{n}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \tag{13}$$

By minimizing the above equation with respect to the parameters $\mathbf{w}$ we could then obtain an analytical expression for the parameters $\mathbf{w}$. Furthermore, a

regularization parameter λ can be added by defining a new cost function to be optimized, that is

$$\mathcal{C}(\mathbf{w}) = \frac{1}{n}((\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})) + \lambda\|\mathbf{w}\|_2^2 \tag{14}$$

which leads to the *Ridge regression minimization problem* where the problem also require that $|\mathbf{w}\|_2^2 \leqslant t$, where t is a finite number larger than zero. Using the matrix-vector expression for Ridge regression by taking the derivatives with respect to $\mathbf{w}$ we obtain then a slightly modified matrix inversion problem:

$$\mathbf{w}^{Ridge} = (\mathbf{X^T X} + \lambda\mathbf{I})^{-1}\mathbf{X^T y} \tag{15}$$

with $\mathbf{I}$ being a $p \times p$ identity matrix with the constraint that

$$\sum_{i=1}^{p-1} w_i^2 \leqslant t \tag{16}$$

The parameter λ that we have introduced in the Ridge (and Lasso as well) regression is often called a regularization parameter or shrinkage parameter. It is common to call it a *hyperparameter*. How hyperparameters work will be based on Covariance and Correlation functions in statistical analysis domain . In particular, by changing the parameter λ, the variance of the parameters $\mathbf{w}_i$ is affected.

For better undenstanding, let's remind about the definition of the covariance and the correlation function. Suppose there are defined two vectors $\mathbf{x}$ and $\mathbf{y}$ with n elements each. The covariance matrix $\mathbf{C}$ is defined as:

$$\mathbf{C}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} cov(\mathbf{x}, \mathbf{x}) & cov(\mathbf{x}, \mathbf{y}) \\ cov(\mathbf{y}, \mathbf{x}) & cov(\mathbf{y}, \mathbf{y}) \end{bmatrix} \tag{17}$$

where $cov(\mathbf{x}, \mathbf{y})$ is defined as $cov(\mathbf{x}, \mathbf{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \overline{x})(y_i - \overline{y})$ with $\bar{t}$ is the mean of $\mathbf{t}$. In probability theory and statistics, **covariance** is a measure of the *joint variability of two random variables* . If the greater values of one variable mainly correspond with the greater values of the other variable, and the same holds for the smaller values (i.e. the variables tend to show similar behavior), the covariance is *positive*. In the opposite case, when the greater values of one variable mainly correspond to the smaller values of the other, (i.e. the variables tend to show opposite behavior), the covariance is *negative*. With this definition and recalling the way to define the variance:

$$var(\mathbf{x}) = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \overline{x})^2 \tag{18}$$

Therefore, we can rewrite the covariance matrix as

$$\mathbf{C}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} var(\mathbf{x}) & cov(\mathbf{x}, \mathbf{y}) \\ cov(\mathbf{y}, \mathbf{x}) & var(\mathbf{y}) \end{bmatrix} \tag{19}$$

The *covariance* takes values between zero and infinity and may thus lead to problems with loss of numerical precision for particularly large values. It is common to scale the covariance matrix by introducing instead the *correlation matrix* defined via the so-called correlation function

$$corr(\mathbf{x},\mathbf{y}) = \frac{cov(\mathbf{x},\mathbf{y})}{\sqrt{var(\mathbf{x})var(\mathbf{y})}} \tag{20}$$

The correlation function has the values $corr(\mathbf{x},\mathbf{y}) \in [1,1]$. This avoids the problems with too large values. Then defining the correlation matrix for the two vectors x and y as

$$K(\mathbf{x},\mathbf{y}) = \begin{bmatrix} 1 & corr(\mathbf{x},\mathbf{y}) \\ corr(\mathbf{y},\mathbf{x}) & 1 \end{bmatrix} \tag{21}$$

In our derivation of the various regression algorithms like *Ordinary Least Squares* or *Ridge regression*, the *design matrix* $\mathbf{X}$ is defined as

$$\mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & ... & x_{0,p-1} \\ x_{1,0} & x_{1,1} & ... & x_{1,p-1} \\ ... & ... & ... & ... \\ x_{N-1,0} & x_{N-1,1} & ... & x_{N-1,p-1} \end{bmatrix} \tag{22}$$

with $\mathbf{X} \in R^{n\times p}$. It can be rewrited in terms of its column vectors as

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_0} & \mathbf{x_1} & ... & \mathbf{x_{p-1}} \end{bmatrix}$$

with a given vector $\mathbf{x_i^T} = \begin{bmatrix} x_{0,i} & x_{1,i} & ... & x_{N-1,i} \end{bmatrix}$ With these definitions, let's rewrite our 2×2 correaltion and covariance matrix in terms of a general *design matrix* $\mathbf{X} \in R^{n\times p}$. This leads to a $p \times p$ covariance matrix for the vectors $\mathbf{x_i}$ with $i = 0, 1, \ldots, p-1$

$$\mathbf{X} = \begin{bmatrix} var(\mathbf{x_0}) & cov(\mathbf{x_0},\mathbf{x_1}) & ... & cov(\mathbf{x_0},\mathbf{x_{p-1}}) \\ cov(\mathbf{x_1},\mathbf{x_0}) & var(\mathbf{x_1}) & ... & cov(\mathbf{x_1},\mathbf{x_{p-1}}) \\ ... & ... & ... & ... \\ cov(\mathbf{x_{p-1}},\mathbf{x_0}) & cov(\mathbf{x_{p-1}},\mathbf{x_1}) & ... & var(\mathbf{x_{p-1}}) \end{bmatrix} \tag{23}$$

and the correlation matrix

$$\mathbf{X} = \begin{bmatrix} 1 & corr(\mathbf{x_0},\mathbf{x_1}) & ... & corr(\mathbf{x_0},\mathbf{x_{p-1}}) \\ corr(\mathbf{x_1},\mathbf{x_0}) & 1 & ... & corr(\mathbf{x_1},\mathbf{x_{p-1}}) \\ ... & ... & ... & ... \\ corr(\mathbf{x_{p-1}},\mathbf{x_0}) & corr(\mathbf{x_{p-1}},\mathbf{x_1}) & ... & 1 \end{bmatrix} \tag{24}$$

Correlation coefficients are also used to measure how strong a relationship is between two variables. Correlation coefficients is a shrinked version from covariance matrix but fully shows the relationship between 2 attributes (2 variables). There are several types of correlation coefficient, but the most popular

is **Pearson's**. Pearson's correlation is a correlation coefficient commonly used in *linear regression*, especially *rigde regression*.

Ridge Regression is a technique for analyzing multiple regression data that suffer from *multicollinearity*. Multicollinearity, or collinearity, is the existence of near-linear relationships among the independent variables. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable. So, **ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity**. [5]

1.1.4 Lasso Regression

In Lasso Regression, the cost function is

$$\mathcal{C}(\mathbf{w}) = \frac{1}{n}((\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})) + \lambda\|\mathbf{w}\|_1 \tag{25}$$

Here is the definition of the norm-1:

$$\|x\|_1 = \sum_{i=0}^{k} |x_i| \tag{26}$$

Lasso stands for *least absolute shrinkage and selection operator*. Lasso regression is also a type of linear regression that uses *shrinkage*. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of *muticollinearity* or used for variable selection (parameter elimination). Lasso regression performs *L1 regularization*, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients: Some coefficients can become zero and eliminated from the model. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models. On the other hand, *L2 regularization* (used in Ridge regression) doesn't result in elimination of coefficients or sparse models. This makes the Lasso far easier to interpret than the Ridge. Which is the same as minimizing the sum of squares, we add the constraint

$$\sum_{i=0}^{p-1} |x_i| \leqslant t \tag{27}$$

some coefficient of $\mathbf{w}$ the are shrunk to exactly zero, resulting in a regression model that's easier to interpret. *A tuning parameter*, λ controls the strength of the L1 penalty. λ is basically the amount of shrinkage:

- When $\lambda = 0$, no parameters are eliminated. The estimate is equal to the one found with linear regression.

- As λ increases, more and more coefficients are set to zero and eliminated (theoretically, when $\lambda = \propto$, all coefficients are eliminated).
- As λ increases, bias increases.
- As λ decreases, variance increases.

For better understanding, *Bias* is the difference between the predicted average of the model we build with the exact value that is trying to predict. A model with a high bias value means that the model does not care much about training data, making it too simple. So it often leads to the fact that the model has a high level of errors both on training and testing exercises. *Variance* characterizes the degree of dissipation of the predicted value for the data point. The high-variance model means it focuses a lot of attention on training data and does not bring generality on data that has never been encountered before. This resulted in the model achieving extremely good results on the training data set, but the results were very bad for the test data set (*overfitting*).

1.1.5 Metric for evaluation

With linear regression models, two methods can be used to assess if a model is considered successful or not. They are **R2 Score** and **RMSE score**. The coefficent $R2$ is defined as:

$$R2 = 1 - \frac{u}{v} \tag{28}$$

where u is the residual sum of squares $((y_{true} - y_{predict})^2).sum()$ and v is the total sum $((y_{true} - y_{true}.mean())^2).sum()$. The best possible score is 1.0 and it can be negative if the model is actually worse. For its value:

- R2 of 0 means that the dependent variable cannot be predicted from the independent variables.
- R2 of 1 means that the dependent variable can be predicted without error from the independent variable.
- An R2 between 0 and 1 indicates the extent to which the dependent variable is predictable.

The *RMSE* indicates the dispersion of the predicted values from the actual values

$$RMSE = \sqrt{\sum_{i=0}^{N-1} \frac{(y - \hat{y})^2}{N - 2}} \tag{29}$$

The smaller the RMSE score, the more it shows that the difference between the predicted output value and the actual output value is not large, which means that the model is much better.

1.2 Linear regression with dataset KAG energy appliance

1.2.1 Dataset

This is experimental data used to create regression models of appliances energy use in a low energy building. The data set is at 10 min for about 4.5 months. The house temperature and humidity conditions were monitored with a *ZigBee* wireless sensor network. Each wireless node transmitted the temperature and humidity conditions around 3.3 min. Then, the wireless data was averaged for 10 minutes periods. The energy data was logged every 10 minutes with m-bus energy meters. Weather from the nearest airport weather station *(Chievres Airport, Belgium)* was downloaded from a public data set from Reliable Prognosis , and merged together with the experimental data sets using the date and time column. Two random variables have been included in the data set for testing the regression models and to filter out non predictive attributes (parameters). Data used include measurements of temperature and humidity sensors from a wireless network, weather from a nearby airport station and recorded energy use of lighting fixtures.

Some specifications about this dataset:

- **date**: time year-month-day hour : minute : second
- $\mathbf{T_1}$: Temperature in kitchen area, in Celsius
- $\mathbf{RH_1}$: Humidity in kitchen area, in percentage
- $\mathbf{T_2}$: Temperature in living room area, in Celsius
- $\mathbf{RH_2}$: Humidity in living room area, in percentage
- $\mathbf{T_3}$: Temperature in laundry room area
- $\mathbf{RH_3}$: Humidity in laundry room area, in percentage
- $\mathbf{T_4}$: Temperature in office room, in Celsius
- $\mathbf{RH_4}$: Humidity in office room, in percentage
- $\mathbf{T_5}$: Temperature in bathroom, in Celsius
- $\mathbf{RH_5}$: Humidity in bathroom, in percentage
- $\mathbf{T_6}$: Temperature outside the building (north side), in Celsius
- $\mathbf{RH_6}$: Humidity outside the building (north side), in percentage
- $\mathbf{T_7}$: Temperature in ironing room , in Celsius
- $\mathbf{RH_7}$: Humidity in ironing room, in percentage
- $\mathbf{T_8}$: Temperature in teenager room 2, in Celsius
- $\mathbf{RH_8}$: Humidity in teenager room 2, in percentage

- **T_9**: Temperature in parents room, in Celsius
- **RH_9**: Humidity in parents room, in percentage
- **T_out**: Temperature outside (from Chievres weather station), in Celsius
- **Press_mm_Hg**: Air pressure outside (from Chievres weather station), in mmHg
- **RH_out**: Humidity outside (from Chievres weather station), in
- **Windspeed**: (from Chievres weather station), in m/s
- **Visibility**: (from Chievres weather station), in km
- **Tdewpoint**: (from Chievres weather station), Â°C
- **lights**: energy use of light fixtures in the house in Wh
- **rv1**: Random variable 1, nondimensional.
- **rv2**: Random variable 2, nondimensional.

  → *Two random variables have been included in the data set for testing the regression models and to filter out non predictive attributes (parameters).*
- **Appliances**: energy use in Wh

⟹ Observation:

- *Temperature along with humidity level in a room will give idea about human prese in the room and hence its impact on Appliance consumption.*
- *The remaining attributes are some weather properties. The weather at that time will also greatly affect energy consumption.*

1.2.2 Reading the data

Some observation about dataset:

- *The number of rows in dataset is* 19735*. The number of columns in dataset is* 29*.*
- *Almost attributes(columns) in this dataset are in float datatype. The rest, Application and lights attributes are int datatype, date is time, object datatype.*
- *No null value is found in the dataset. This dataset can be evaluated as complete.*

About the range of attributes:

- **Temperature columns** : Temperature inside the house varies between 14.89 Deg 29.85 Deg , temperatire outside (T_6) varies between -6.06 Deg to 28.29 Deg . The reason for this variation is sensors are kept outside the house.
- **Humidiy columns** : Humidity inside house varies is between 20.60% to 63.36% with exception of RH_5 (Bathroom) and RH_6 (Outside house) which varies between 29.82% to 96.32% and 1% to 99.9% respectively.
- **Appliances** - 75% of Appliance consumption is less than 100 Wh . But the maximum consumption of 1080 Wh , The distribution of this column will be uneven (There are small number of cases where consumption is very high).
- **Lights column** - With 11438 0 (zero) enteries in 14801 rows , this column will not add any value to the model . Hence, maybe drop this column. But through many experiments, removing the light column will reduce the score so that after all, not drop this column.

1.2.3 Data Visualization

Here is a histogram to observe the value distributions of the dataset's attributes to evaluate the type of distribution as well as detect the abnormalities of the data. For better understanding, suppose a random variable X with a certain density of distribution. We get N samples $X(1), X(2), ..., X(N)$ where X varies from $max(X)$ to $min(X)$. when using the $hist(X, bin)$ function, it will divide the interval $[min(X), max(X)]$ into 'bin' segments and count in each segment $[n * (max(X) - min(X))/bin, (n+1) * (max(X) - min(X))/bin)]$ how many X values appear and then represented as a column of each bin.

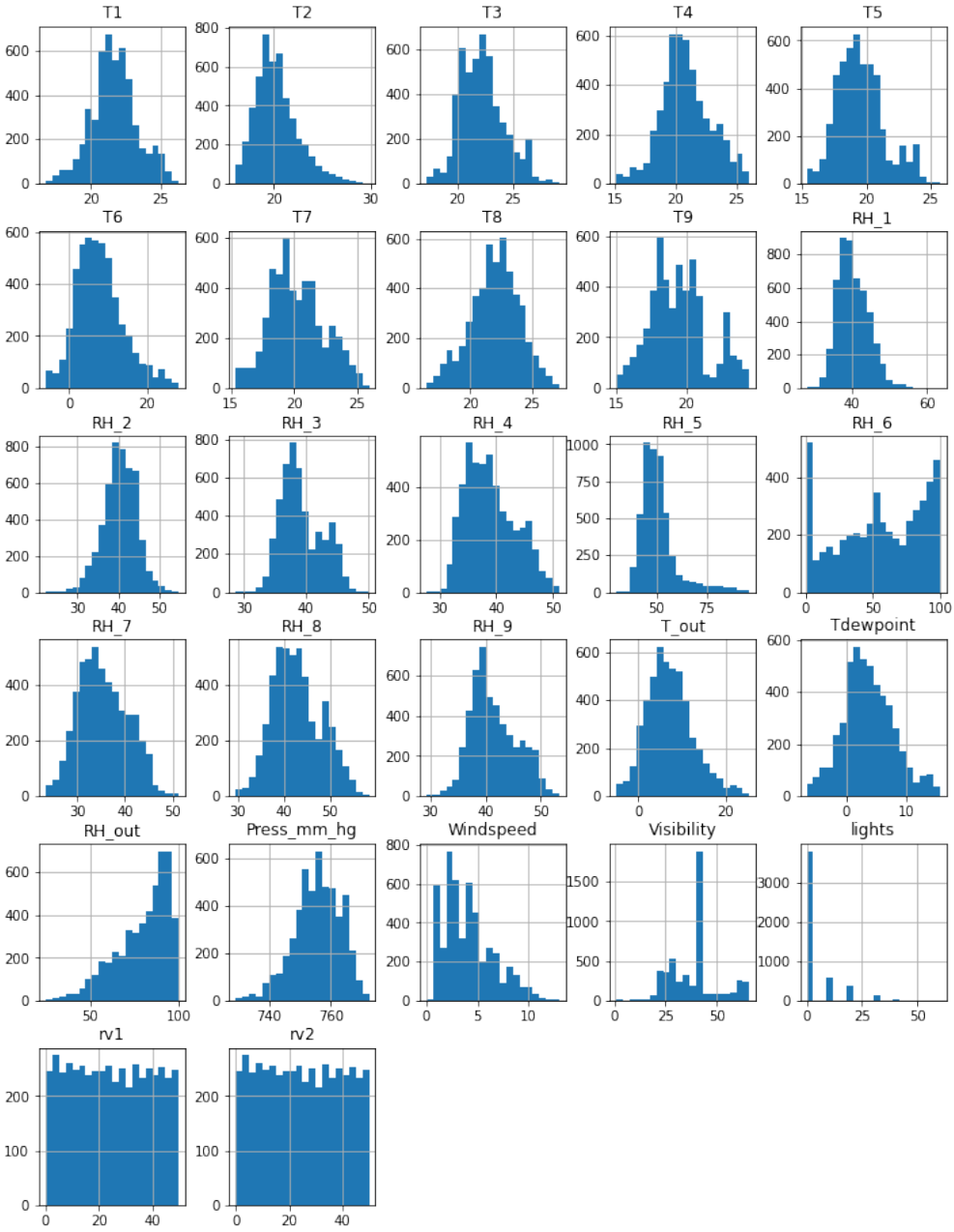

Figure 1: *The Histogram of each input figure in the dataset*

The more general view about the distribution of attributes can be found on **Figure 1**. Data of attributess mostly distributed in an acceptable distribu-

tion(*i.e.* in the form of normal distribution). Beside that, it is easy to notice that histograms for RH_6 , RH_out , Visibility , Windspeed, T_9 and lights columns have irregular distributions. r1 and r2 are random variables have the uniform distribution.

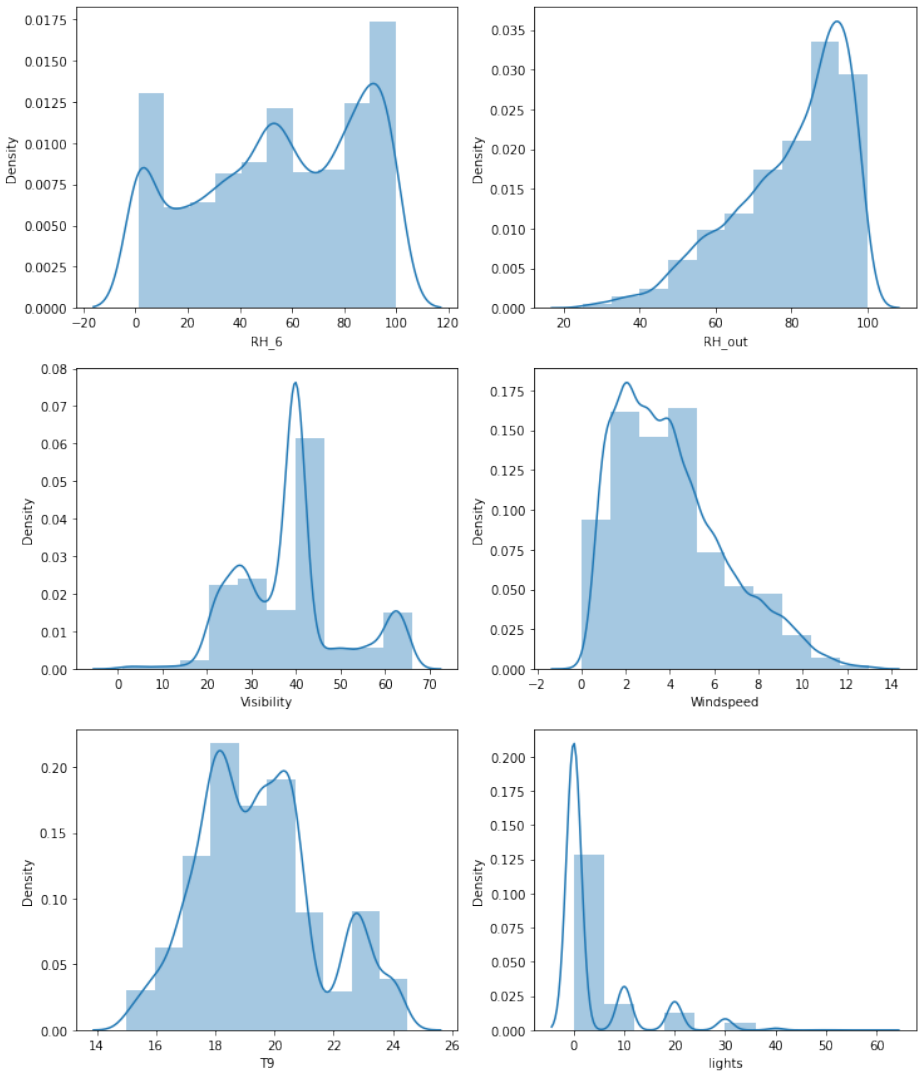

Figure 2: *The Displot of RH_6 , RH_out , Visibility , Windspeed, T_9 and lights columns in the dataset*

Through **Figure 1** and **Figure 2** we have some observations :

- **Temperature** : All the columns follow normal distribution except T9 have mix distribution
- **Humidity** : All columns follow normal distribution except RH_6 and RH_out , primarly because these sensors are outside the house. The weather is erratic and usually the indoor climate is more stable.
- **Visibilty** : This column is negatively skewed
- **Windspeed** : This column is postively skewed
- At the same time, the **lights** column has at least 75% value 0. Maybe this is an unimportant attribute during the training because it contributes almost nothing to the linear regression.

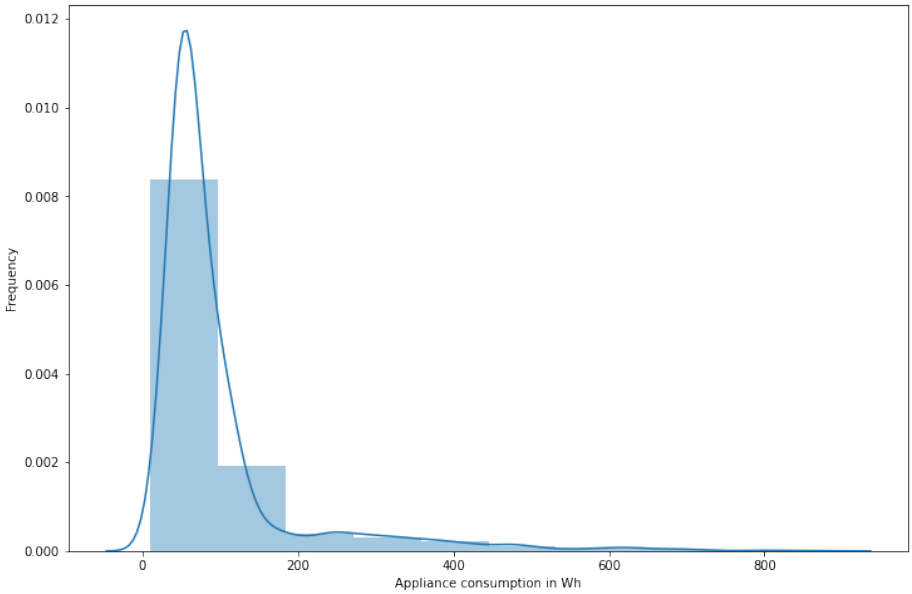

Figure 3: *The Displot of Appliances column in the dataset*

Through **Figure 3**, the Appliances column is postively skewed , most the values are around mean 100 Wh . There are a lot of outliers in this column. For better understanding, the **mean** (average) of a data set is found by adding all numbers in the data set and then dividing by the number of values in the set. The **median** is the middle value when a data set is ordered from least to greatest. The **mode** is the number that occurs most often in a data set. If the distribution has **a positive skew**, then $mean > median > mode$. If the distribution has **a negative skew**, then $mean < median < mode$.

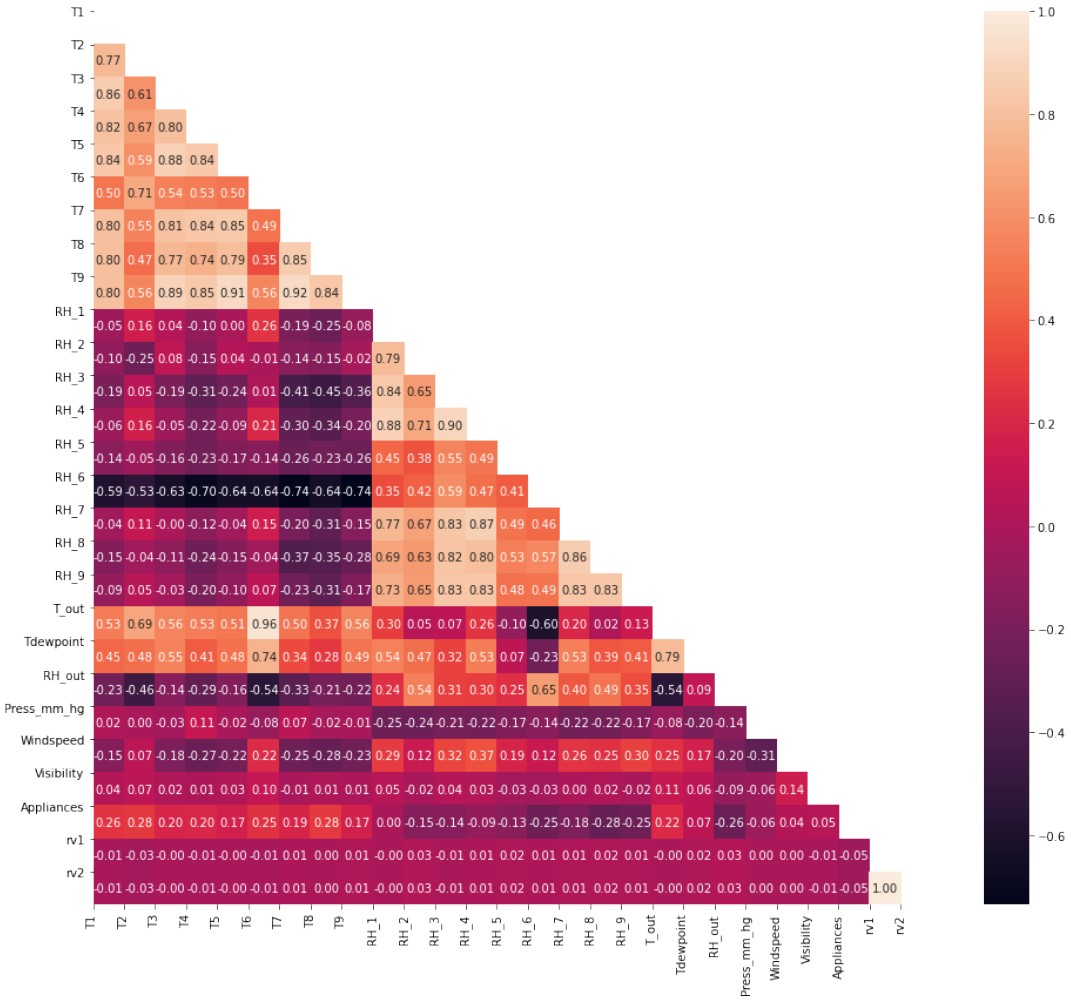

Figure 4: *The Correlation Matrix of figures in the dataset*

Through **Figure 4** we have some observations :

- **Temperature** : All the temperature variables from $T_1 - T_9$ and T_out have positive correlation with the target Appliances . For the indoor - temperatures, the correlations are high, since the ventilation is driven by the HRV unit and minimizes air temperature differences between rooms. Four columns have a high degree of correlation with T_9 are T_3, T_5, T_7, T_8. T_6 and T_out has high correlation maybe both temperatures from outside so they are the same.
- **Humidity** : There is significantly high correlation case for humidity sensor RH_4 with RH_3, RH_1, RH_7 and other sensor.

- **Weather attributes** : Visibility, Tdewpoint, Press_mm_hg have low correlation values . But Visibility is almost uninvolved with every attributes.
- **Random variables**: two variables have no role to play with other attributes (almost 0.0 correlation with other attributes) but have the absolute link between them (1.0 for correlation). This can be explained by that these two variables are added by humans for the purpose of replacing the missing attributes that have an impact on the output.

1.2.4 Outlier Detection

Now let's plot the **boxplot** to identify the outliers that exist in the features. When a data set is ordered from least to greatest, $Q1$ is the 25% value, the median is the middle value, $Q3$ is the 75% value. The value IQR is calculated by $IQR = Q3 - Q1$. Then from the Q1 value down, exceeding $1.5IQR$ from $Q1$ means **outlier**. Similarly, from the $Q3$ value, exceeding $1.5IQR$ from $Q3$ means outlier.

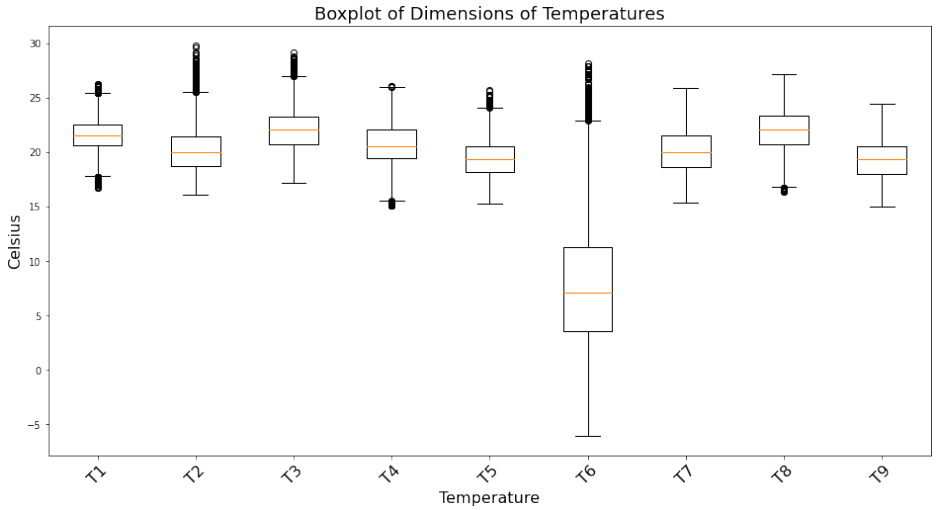

Figure 5: *The Boxplot of Temperature figures in the dataset*

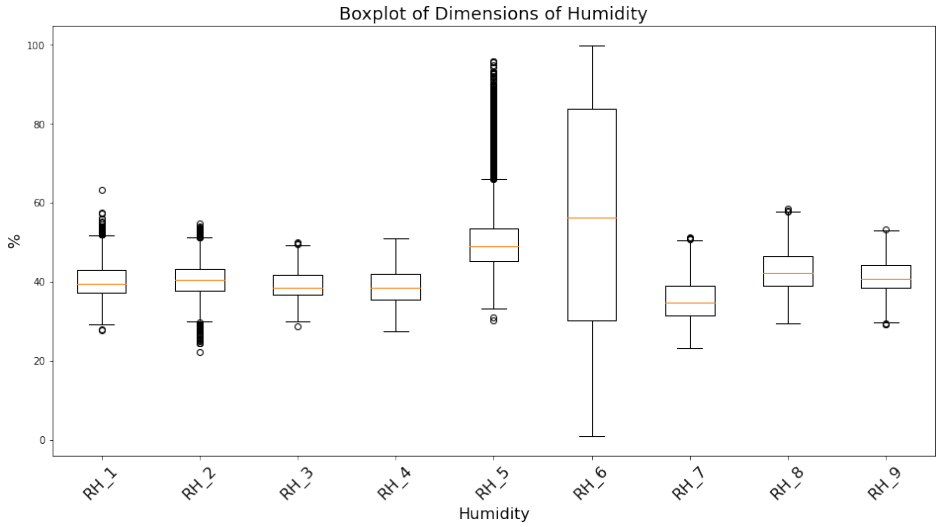

Figure 6: *The Boxplot of Humidity figures in the dataset*

Through **Figure 5** and **Figure 6** we have some observations :

- **Temperature** : T_6 has the a big range of values. Almost of columns have few outliers, except T_2 and T_6 have a lot.
- **Humidity** : RH_6 also has the a big range of values. Almost of columns have few outliers, except RH_2 and RH_5 have a lot
- **lights** : The range of value is too small. Almost of values is 0 so it does not have too much outliers.
- **Weather attributes** : Although their boxplots were not shown here because the weather attributes have different units of measurement. However, there are a few points we can note:
  - **Visibilty** : This column is negatively skewed. This column has lots of outliers which higher or lower than permitted value.
  - **Windspeed** : This column is postively skewed. This column has few outliers.
  - **Press_mm_hg** : There were a lot of outliers lower than normal value.

  this can be explained by the fact that weather figures often change abnormally leading to many outliers.

Finally, let's use the boxplot model to *eliminate outliers*. It means that the entire data point will be eliminated if at least one value of it is considered as outlier in a specific column. Through many experiments, we have determined

that in this linear regression model, eliminating outliers in all figures gives the best results. This can be reasoned by limiting the overwhelming transformation of component figures and results. That creates data focus and increases performance. For specification, if we does not do anything, the result of the *R2 metric* is only 13.47% . When only eliminating the outliers in attribute 'appliances' (the result) of the train set, the result of the *R2 metric* plummeted, which can be explained by that we have made the train data more concentrated but the test set data is still the same resulting in the above result. When eliminating outliers of some attributes that have lots of outliers in train test, such as visibility,.. the result increased but it's not significantly, *R2 metric* is about 15.84%. However, when eliminating outliers in all figures and in both train and test sets combined with some other methods, *R2 metric* results have reached 34.45%.

1.2.5 Linear Regression Implementation

In a linear regression problem in particular and machine learning in general, the ratio of training and testing will affect the results of the assessment. However, what we expect is that the train training rate will be as small as possible while at the same time being able to give an appreciated result. Through many experiments, for this linear regression model, we dicide that the test rate is 75% for the highest result. We perform training the train set after being removed outlier with the original linear regression model and 2 improved versions Ridge regression and Lasso Regression. We then evaluate the results using **R2** and **RMSE** metrics.

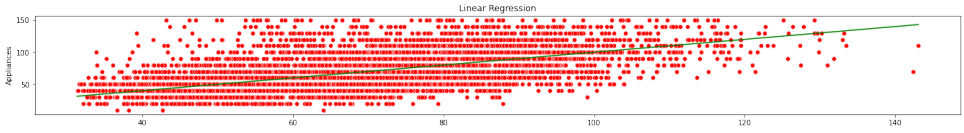

Figure 7: *Linear Regression line of original Linear Regression*

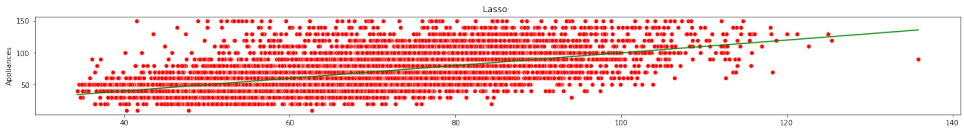

Figure 8: *Linear Regression line of Ridge Regression*

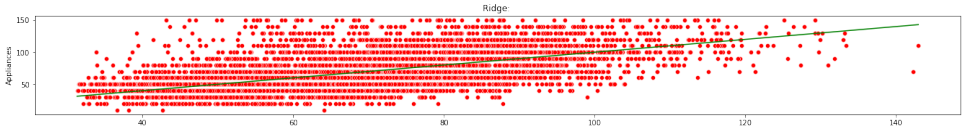

Figure 9: *Linear Regression line of Lasso Regression*

On **Figure 7**, **Figure 8** and **Figure 9**, the green lines show predictive values while red points show actual values (y-axis) based on predictive values (x-axis).

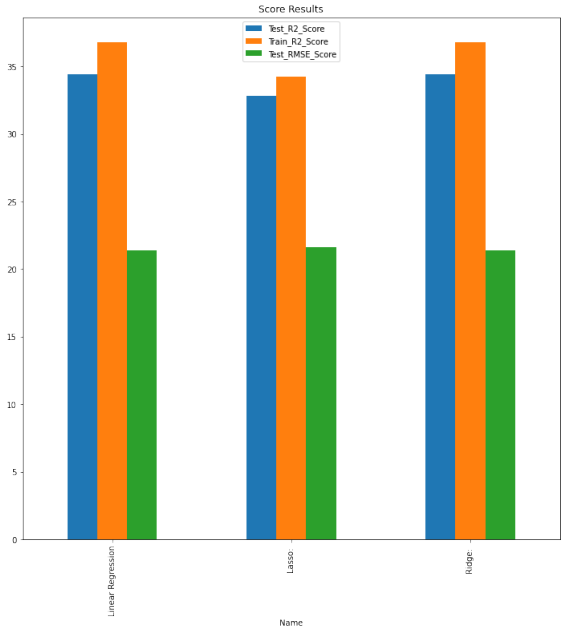

Figure 10: *Linear Regression Score*

On **Figure 10**, the blue bar show the R2 Score of the model on the test set (in the 100× unit). The orange bar show the R2 Score of the model on the train set (in the 100× unit). The green bar show the RMSE Score of the model on the test set.

Conclusion: After carrying out many experiments, these conclusions have been drawed:

- The accuracy of linear regression has increased significantly thanks to the removal of outliers (from 0.14 to 0.34).
- The elimination of outliers needs to be done in both training and testing of data to achieve the highest efficiency.

- Best results over test set are given by Ridge Regression with **R2 score** of 0.3450.
- Least **RMSE score** is also by Ridge Regression 21.3705.

Standardization has also been applied to this model for the purpose of increasing performance, for the following results.

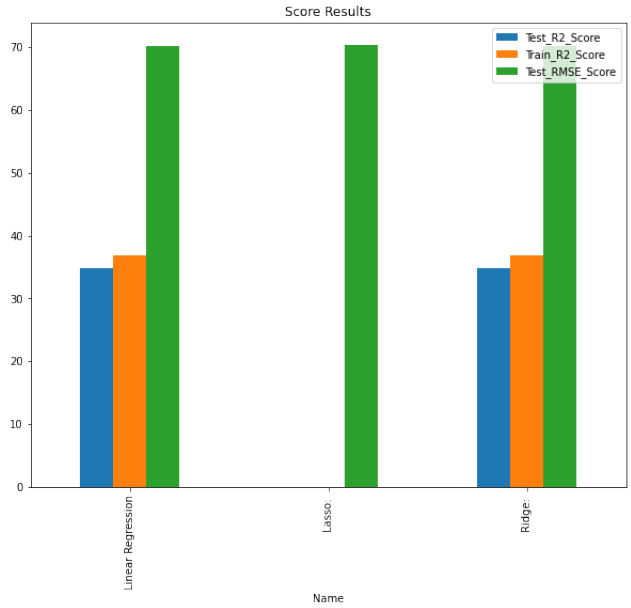

Figure 11: *Linear Regression Score after Normalization*

Observation:

- The R2 Score has increased but not significantly with the original linear regression and Ridge regression.
- For Lasso Regression, the R2 Score is 0 for the train test and the test set. This can be explained by the fact that when we standardize the data, this has caused the vector **w** parameters to be shirked to 0 almost entirely resulting in the output being almost a constant. R2 equals 0 which means that this lasso model *cannot predict the output value based on the given inputs*.

  $\Longrightarrow$ When modeling linear regression using Lasso regression algorithm, we should not standardize the data

Feature selection has also been applied to this model for the purpose of data concentration and reducing the training time. After evaluating the importance of data properties, we remove the unimportant features and retain

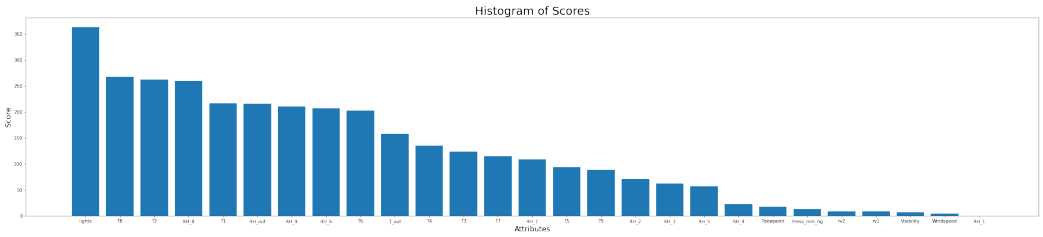

Figure 12: *The Importance of features using f_regression method for estimation*

the important features in the creation of output. Through many experiments, we see that removing attributes when using **f_regression** method to evaluate attributes almost only reduces the score of the model down. For example, when we retain 20 properties that use f_regression as an evaluation method, the time for training is reduced but R2 Score is only about 0.315 (reduced from 0.34) and RMSE also increases. Histogram in **Figure 13** a histogram that shows the im-

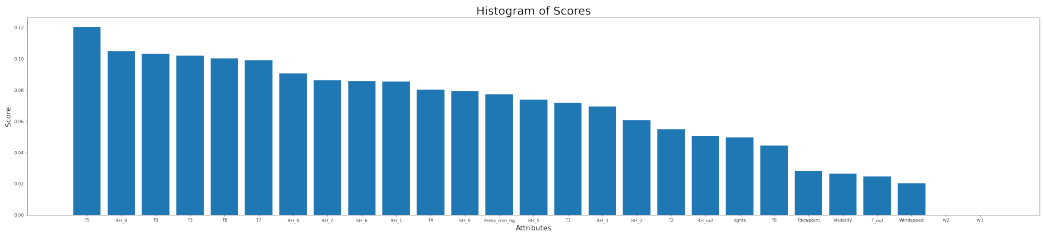

Figure 13: *The Importance of features using mutual_info_regression method for estimation*

portance of figures in the creation of output using the **mutal_infor_regression** method. We can see that the scale and the scores are different between the two methods. After evaluation and elimination some unimportant figures, we have conducted an inspection of the model's result value. The results showed that R2 scores had fallen (but not significant, is about 0.34) while RMSE had increased but compared to f_regression, this result was somewhat better.

Observation:

- The removal of some features does not increase or significantly increase the R2 Score. Indeed, it reduce the scores.
- Use the *mutual_info_regression* function to evaluate the scores for features and then eliminate the low-score features that achieve higher efficiency than the *f_regression*.

Conclusion:

- The best Algorithm to use for this dataset is Ridge.

- The final model (the best-score model) had unchanged features with the origin.
- Feature reduction was not able to add to better R2 score

Finally, here are some of the results we got during our testing, they are R2_Score in test set:

	test set size = 0.25, not remove outliers, not remove attributes, no normalization	test set size = 0.25, remove outliers in *Appliances* column, remove *lights* column, no normalization	test set size = 0.25, remove outliers in *Appliances* column, remove *lights, rv1, rv2, Visibility, T6, T9* column, no normalization	test set size = 0.25, remove outliers in *Appliances* column, remove *lights, rv1, rv2, Visibility, T6, T9* column, normalization	test set size = 0.5, not remove outliers, not remove attributes, no normalization
Linear Regression	0.13476	0.13476	0.12156	0.12136	0.13801
Lasso Regression	0.13468	0.13469	0.12272	0.0000	0.13727
Ridge Regression	0.13477	0.13477	0.12157	0.12139	0.13801

Table 1: *Table of some results with many different versions*

test set size = 0.5, not remove outliers, remove *Visibility* column, no normalization	test set size = 0.75, remove outliers in *Visibility* column, not remove atributes, no normalization	test set size = 0.75, remove outliers in all columns in train and test set, not remove attributes, no normalization	test set size = 0.75, remove outliers in all columns in train and test set, not remove atributes, normalization	test set size = 0.75, remove outliers in all columns in train and test set, retain 20 important attributes based on *f_regression*	test set size = 0.75, remove outliers in all columns in train and test set, retain 20 important attributes based on *mutual_info_regression*
0.13801	0.15849	**0.34447**	**0.34817**	0.31614	**0.34077**
0.13727	0.15755	**0.32808**	0.0222	0.29865	**0.32400**
0.13801	0.15850	**0.34450**	**0.34828**	0.31462	**0.34078**

Table 2: *Table of some results with many different versions (continue)*

1.3 Linear regression with dataset California Housing Prices

Linear regression is one of the most basic supervised learning algorithms in machine learning. In this task, we have used linear regression in order to predict

prices of houses in California. The rest of this part of the report is organized as follows. 1.2 provides details about the dataset. Next, we will explore data and visualize histogram, distribution in section 1.3. Then we will use model Linear regression in python library Sklearn in section 1.4. Next, section 1.5 provides one way to remove a little noises by standard score step by step.

1.3.1 Dataset

This dataset includes information about houses in California derived from the 1990 census. The dataset is uploaded to Kaggle, a website about machine learning for everyone who wants to learn machine learning. And **Table 3** will show details of this dataset:

	longitude	latitude	house age	total rooms	total bedrooms	population	households	median income	median house value
count	17000	17000	17000	17000	17000	17000	17000	17000	17000
mean	-119.56	35.62	28.58	2643.66	539.41	1429.57	501.22	3.88	207300.91
std	2.00	2.13	12.58	2179.94	421.49	1147.85	384.52	1.90	115983.76
min	-124.35	32.54	1.00	2.00	1.00	3.00	1.00	0.49	14999.00
max	-114.31	41.95	52.00	37937.00	6445.00	35682.00	6082.00	15.00	500001.00

Table 3: California dataset details.

Following the **Table 3** show the number of samples, mean, standard deviation and range of each feature in dataset.

1.3.2 Data visualization

First, we use information about geographical location, median incomes and median prices to imagine effects of location and median income to values of California houses in Figure 14. Obviously, which houses locate near sea and center have high value than others, the right sidebar explains color that corresponds to the value of the house. And the size of circles, is corresponding with median income of the house.

And next we will consider the histogram of median house value, which feature we will predict in Figure 15.

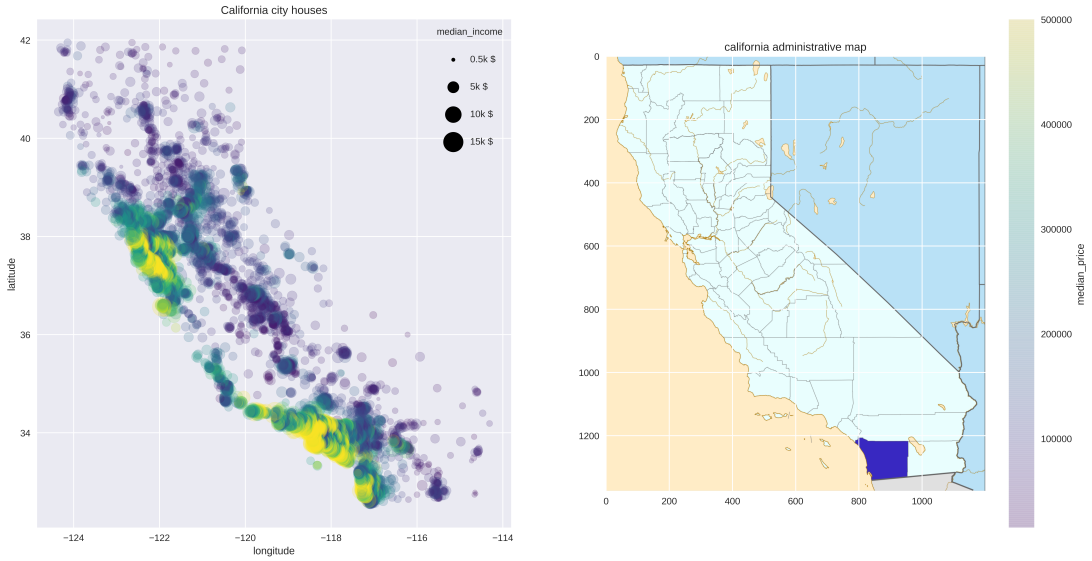

Figure 14: Location and median incomes, median prices

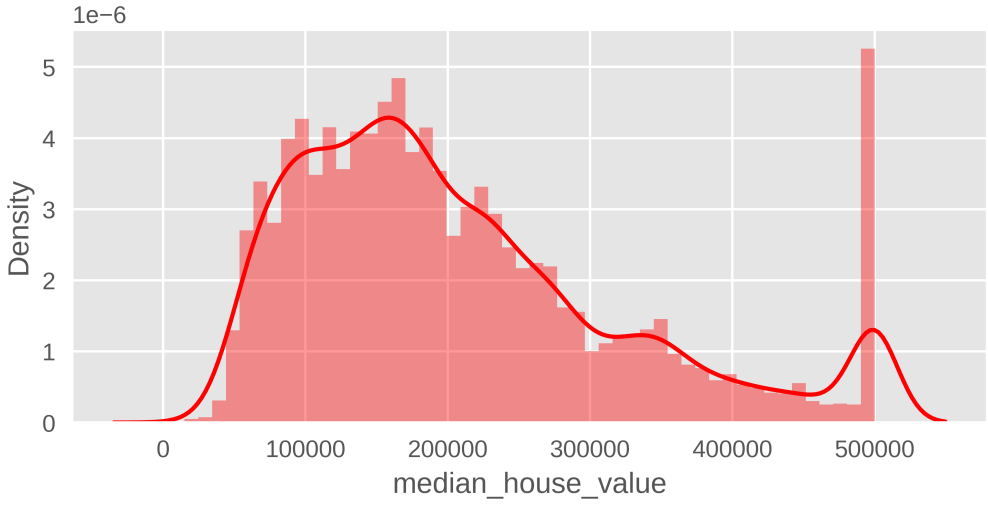

Figure 15: Histogram of median house value

The distribution of median house income is similar to normal distribution, but on the right of plot, houses have value 50000 is extraordinary, so these houses may be outliers in our dataset.
Normalize data to range [0-1], with x_i is a i^{th} feature of one point $\mathbf{x}$. We have

the formula:

$$z_i = \frac{x_i - min(k_i)}{max(k_i) - min(k_i)} \tag{30}$$

where k_i is all values of k_i^{th} feature in dataset. Use boxplot to see how data distributes and noises in our dataset.

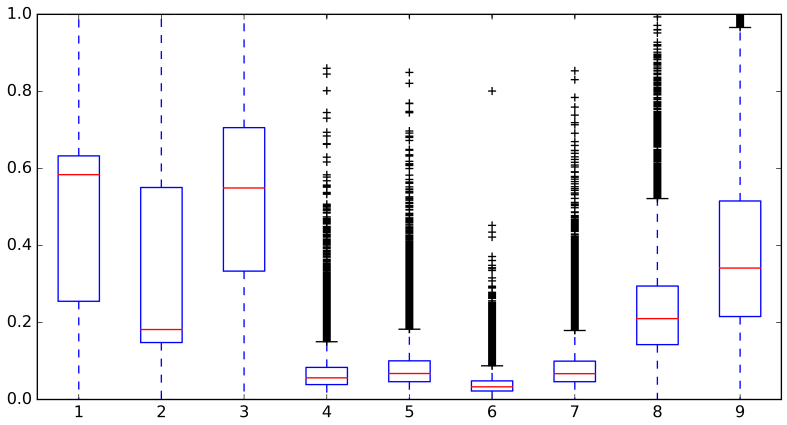

Figure 16: Boxplot data visualization

1.3.3 Linear regression with Sklearn

After apply the model to the dataset with test size = 0.3, the mean square error (MSE) is 0.02121. And sklearn.linearregression.score reachs 0.6305.

1.3.4 Remove noises by standard scrore

By figure 16, we can see that there are to much outliers in our dataset, so we will use standard score to improve noises by steps follows:

Step 1: Calculate standard score for each point
We will split median house price into 10 intervals range, which range, we will calculate the mean, standard deviation, and standard score is norm 2 (Euclid distance) of each point on local standard deviation unit.

$$Standardscore = \left\| \frac{x_i - \mu_i}{\sigma_i} \right\|_2 \tag{31}$$

And below is histogram figure of standard score:

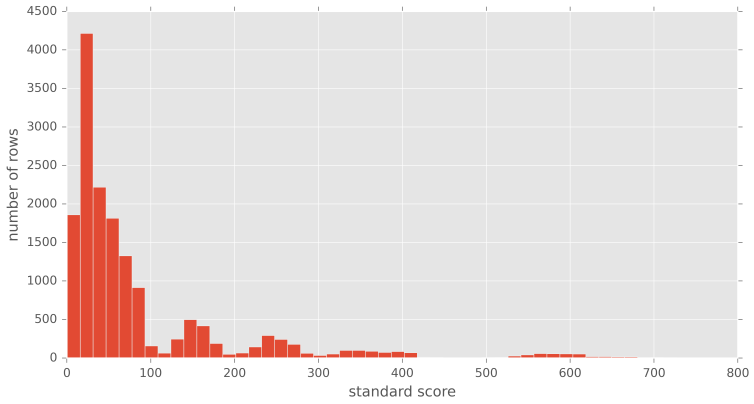

Figure 17: Histogram of median house value

High probability which points have standard scores exceed 500 can be ouliers, so we will remove them.
Step 2: Evaluate result
MSE is 0.0198 and sklearn.linearregresion.score reachs 0.6591.Through this, it can be seen that the effect of selecting appropriate features and removing noise contributes to the accuracy and efficiency of linear models in general and linear regression in particular.

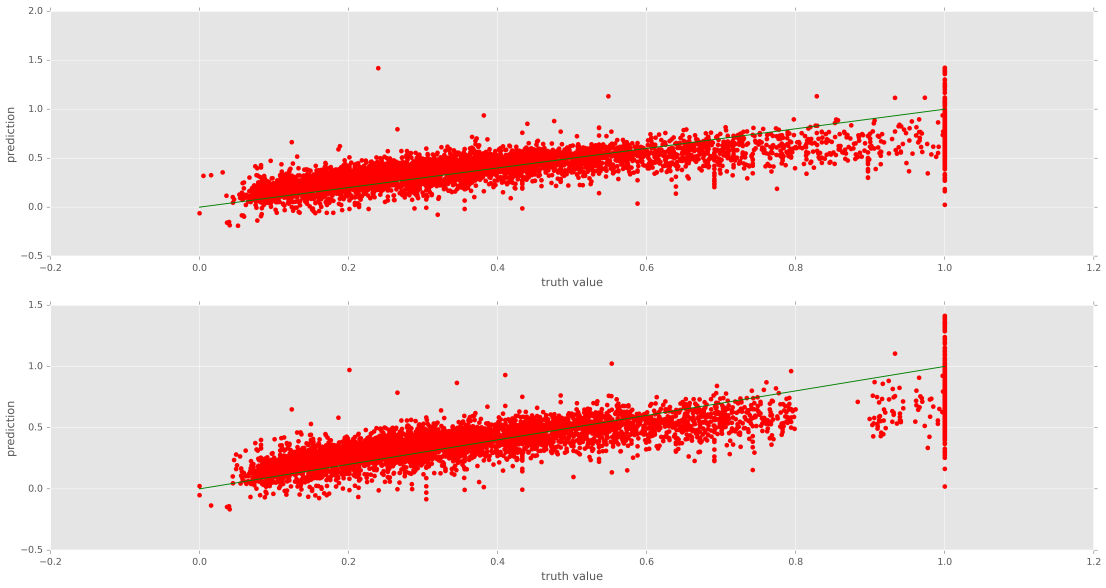

Figure 18: Compare 2 results

On figure 5, horizontal axis is truth value, vertical axis is prediction price of models, points has big distance to green line maybe outliers.

1.4 Summary

Result of linear regression model depends on many different factors of data as data dispersion, distribution of each feature, outliers,... We have implemented this model on datasets of different sizes, with different distributions, different amounts of noise to analysis and evaluate. The results are recorded in the following table.

Dataset name	Feature	Instance	Score1	MSE1	Score2	MSE2	Outlier
KAG energy data	28	19735	0.3444	0.0077	**0.3481**	**0.0022**	5694
California Housing Price	9	17000	0.6305	0.0212	**0.6447**	**0.0196**	389
GPU performance	15	241600	0.4057	0.0072	**0.4716**	**0.0058**	69393
Temperature estimation	8	7588	0.7658	0.0046	**0.7735**	**0.0045**	182
Physicochemical Properties of Protein Tertiary Structure	9	45730	0.2865	0.0609	**0.2924**	**0.0601**	100

Table 4: *Table of some results for 5 datasets*

In table 4, Score 1 and Score 2 is *Sklearn.linear_model.Linear_regression.score* which we have introduced before. MSE is mean square error and MSE1, MSE2 are 2 values of model before and after use standard score to remove some noise. Last column is number of outliers that we removed. On all datasets, after use standard score to remove noises, score is better and MSE is lower than before. Therefore, linear regression model is very sensitive with noise, data before input to linear model should be preprocessed to achieve good score.

References

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[2] Morten Hjorth-Jensen. "Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis". In: (Sep 11, 2020). URL: https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html?fbclid=IwAR0Eg1xhBhzZ3mwK4ue9oh6NtaSsWbyx6D9_1gd8PjM4k_s59YKW9DJDqhE.

[3] Tiep Vu Huu. "LinearRegression". In: (Dec 28, 2016). URL: https://machinelearningcoban.com/2016/12/28/linearregression/.

[4] Tiep Vu Huu. "Singular Value Decomposition". In: (Jun 7, 2017). URL: https://machinelearningcoban.com/2017/06/07/svd/.

[5] NCSS Statistical Software. "Ridge Regression". In: (). URL: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf.

2 Data vectorization

2.1 Image data vectorization

2.1.1 Local Binary Pattern (LBP)

LBP (Local Binary Patterns) is the algorithm for extracting texture features on grayscale images proposed by *Ojala et al.* in their 2002 paper. LBP is widely used in image processing, especially in face recognition and other applications. [3] LBPs compute a local representation of texture. This local representation is constructed by comparing each pixel with its surrounding neighborhood of pixels.

The first step in constructing the LBP texture descriptor is to convert the image to grayscale. For color images, we can split the color image into 3 **RGB** channels and then treat each color channel as a grayscale image. The simplest and most classic LBP method is that at each point of the image, *we consider* 8 *points around the point*. Then, subtract each value of the surrounding point from the value of the point in consideration. We also use an activation function called **threshold** to mark the value calculated to the binary pattern. The threshold will be marked as 1 if the value of the surrounding point greater than or equal the value of the point in consideration and if less than threshold will be marked as 0. Then the values after calculating the threshold (0 or 1) will be multiplied by **the matrix weighted** (we can multiply clockwise or counter-clockwise) and used to calculate the LBP value of the point under consideration in the result matrix.

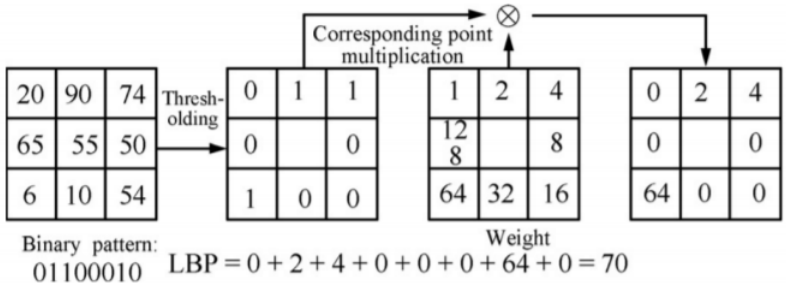

Figure 19: *The second step in constructing a original LBP is to take the 8 pixel neighborhood surrounding a center pixel and threshold it to construct a set of 8 binary digits for calculate the LBP value.*

Repeat the above process through the entire pixels of the images, we will have the output a result matrix equal to the size of the input image. Each value on the output image is an LBP feature. The last step is to compute a histogram over the output LBP array (matrix). For the original LBP algorithm, when consider a 3×3 neighborhood matrix, we has $2^8 = 256$ possible patterns, our LBP **2D** array thus has a minimum value of 0 and a maximum value of 255,

allowing us to construct a 256-bin histogram of LBP value frequencies as our final feature vector. ⇒ The primary benefit of this original LBP implementation is that we can capture extremely fine-grained details (details in a small area) in the image. But being able to capture details at such a small scale is also a drawback to the algorithm, this method will make the features of LBP is not stable and cannot represent very large features, it means capturing details at varying scales is impossible, only the fixed 3×3 scale at all. However, the biggest drawback to the algorithm is that we combined all the LBP values of all pixels in a histogram. It means although the details in small areas have been calculated, but it is not kept as a part in the LBP output vector. This can cause a reduction in classification performance because the LBP vector does not represent features of small areas, but instead overall features of the whole image. Therefore, the object detection process is not good, the noise can cause a great reduction in the classification results.

To handle this, an extension to the original LBP implementation was proposed by *Ojala et al* to handle variable neighborhood sizes. Accordingly, in a grayscale image, the Neighborhood points are not sampled from the 8 surrounding points but will be *a set points on the circle with the point being considered as the center*. [5] Generalizing the above LBP approach, we will have the following parameters:

- **P**: Number of pixels adjacent to the center pixel to consider. (ex: $P = 8$).
- **R**: Radius of neighboring pixels that we will consider - how many pixels away from the center pixel (eg: $R = 1$ means adjacent).
- The **order** of neighboring pixels encoded into the 8-bit string will be clockwise or counter-clockwise.
- **Interpolation:** Due to taking neighboring pixels in a circle, so the coordinates of neighboring pixels when calculating out will be real numbers. So we need to decide how to get the value of neighboring pixels: nearest pixel **(nearest)** or weighted **(bilinear)**. This interpolation is similar to when you resize an image.

  For better understanding, **Interpolation** is the process used to estimate an image value at a position between two pixels of known value. For example, if we resize an image, which will contain more or less pixels than the original image, the toolbox will use interpolation to calculate the value for the additional pixels. Some methods of image interpolation:

  - **Nearest interpolation**: The value of neighboring pixel will be assigned the value of the pixel containing its corresponding point in the original image.
  - **Bilinear interpolation**: The value of the new pixel is the weighted average of 2×2 neighboring pixels.
  - **Bicubic interpolation**: The value of the new pixel is the weighted average of 4×4 neighboring pixels.

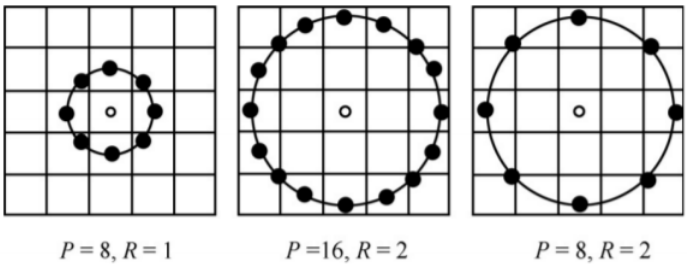

Figure 20: *Three neighborhood examples with varying P and R used to construct Local Binary Patterns.*

Then, the value representing the local texture around the point under consideration will be calculate like before. However, there is an important problem that if the number of neighboring points is considered too large, the characteristic value range for the texture will be too much big. This interferes with feature representation and classification. To solve this problem. Ojala introduced a new concept – **"Uniform patterns"**. A LBP binary sequence is considered to be *uniform* if it has at most two 0-1 or 1-0 transitions. For example, the pattern 00001000 (2 transitions) and 10000000 (1 transition) are both considered to be uniform patterns since they contain at most two 0-1 and 1-0 transitions. The pattern 01010010 on the other hand is not considered a uniform pattern since it has six 0-1 or 1-0 transitions. The number of uniform prototypes in a Local Binary Pattern is completely dependent on the number of points P. As the value of P increases, the dimensionality of your resulting histogram will increase also (because the available LBP value has a big range). Come back to the uniform concept, when the total transitions is less than or equal to 2, the binary sequence is called the *uniform sequence*, their values are calculated as before. The remaining sequences will be called *hybrid sequences* and have the same values. Therefore, the value of the sample representation will be significantly reduced. For example, with 8 neighboring points, the sample representation value is reduced from 256 to 58 (this can be easily proved).

$$\mathbf{LBP}_{P,R}^{\mathbf{modified}} = \begin{cases} \sum_{i=0}^{P-1} s(g_i - g_c)2^i & \textit{for total transitions in binary sequences at most 2} \\ P+1 & \textit{for the remaining cases} \end{cases} \tag{32}$$

where $s(x)$ is the activation function, g_c is the pixel's value for the considered point and g_i is the pixel's value for the neighboring point.

Conclusion:

- **Advantages:**
  - The advantage of the LBP method is that it has *low computational cost*, is *stable* when the intensity of pixels changes linearly, and is

easy to expand into multi-dimensional space such as color images in the RGB system.

- Another advantage of LBP method is that it allows us to *extract features of an image without regard to brightness*, while different brightness affects the distribution of the value of the image pixels, LBP only interested in the internals of the image, which is how the pixels are different from each other (disparity). Therefore, although the same object in different brightness will have different pixel value, in LBP, they will be extremely the same. ⇒ LBP add an extra level of rotation and grayscale invariance,

- **Disadvantages:**
  - The LBP sketches out the intensity distribution around a pixel *in a rather rough way* since it is represented by only two values of 0 and 1. This representation may not be enough for complex features.
  - *Sensitive to interference.* In the problem of recognizing an object, if the noise pre-processing (removing the background) is not good enough, it can lead to erroneous results because the resulting LBP histogram is calculated over the entire input image.

During the experiment, let's used **LeNa_color.png** image to implement LBP algorithm. First, the simplest, let's read the input image file as grayscale image thanks to the *cv2.imread()* library by adjusting the second parameter to 0 after the first parameter is the image address.

Figure 21: *The input image has been resized*

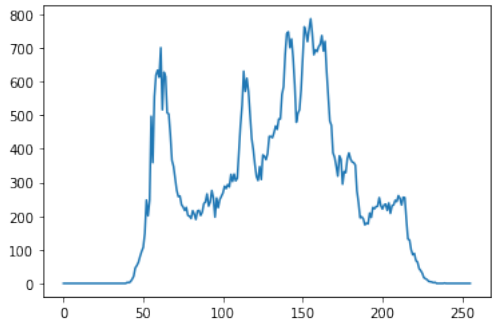

Figure 22: *The histogram of input image*

Figure 23: *The LBP image with $P = 8$ and $R = 1$*

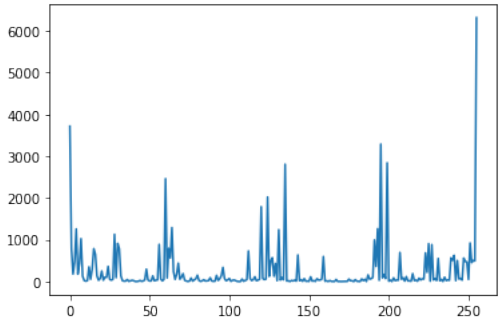

Figure 24: *The LBP histogram with $P = 8$ and $R = 1$*

Figure 25: *The LBP image with $P = 8$ and $R = 2$*

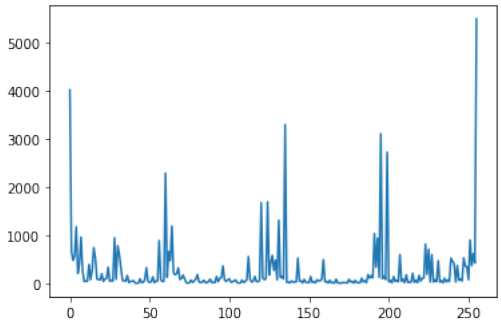

Figure 26: *The LBP histogram with $P = 8$ and $R = 2$*

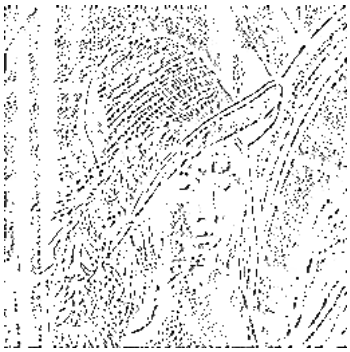

Figure 27: *The LBP image with $P = 16$ and $R = 2$*

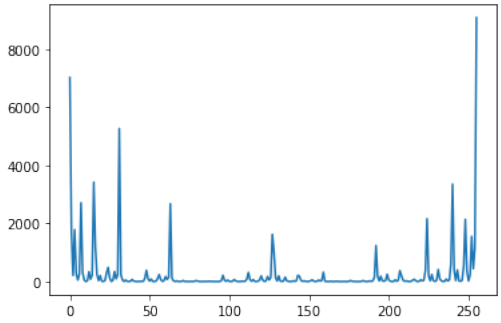

Figure 28: *The LBP histogram with $P = 16$ and $R = 2$*

Observation:

- **Between the original image and resized image:** Although it was not shown here, however, based on the research results, some observations can be made:
  - Before resizing the image to 256×256, the histogram of the original image and that of the resized image have similar shapes. However, the difference that can be seen is that the histogram of the resized image is somewhat more stable and less volatile. It is thanks to the interpolation when we use the resize function to transform the image, which almost keeps the distribution ratio of the pixels.
  - The LBP image of the resized image is somewhat sharper (showing the dividing lines more clearly). In contrast, the LBP image of the original image looks somewhat blurred, the dividing lines in the image are light and not too prominent. $\Rightarrow$ The image resizing is somewhat meant to be a feature extraction as it makes the features of an image clearer when fed into more powerful feature extraction algorithms.
- **Between the resized image and LBP image:**
  - As we can see, the LBP image has a completely different distribution of pixel values than that of the resized image. Instead of focusing in the middle, the histogram of the LBP image is somewhat evenly spaced, concentrated only at some specific values, and most concentrated in the two values 0 and 255. This has shown a characteristic that have been mentioned in the theoretical part above that it is the LBP image that extracts the feature of the image regardless of the brightness, which is also the reason for this difference in distribution. Sure, when increase or decrease the brightness of the input image, the pixel value distribution of the input image will change, but the LBP image will remain the same.
- **Between the LBP's versions:**
  - When R is increased to 2 and retained $P = 8$ (**Figure 23** and **Figure 25**, the LBP image looks sharper and the separation lines are more noticeable. Regarding the histograms of these two versions, we see that the shapes are quite similar, only differing in a few notable columns. This can be explained that taking the same 8 neighboring points to consider although with the changing circle size (keeping the interpolation method unchanged) does not change too much to the LBP histogram of the image (**Figure 24** and **Figure 26**). However, by doing so, we can capture more complex features when considering the wider surrounding space, that's why we see the difference between the two LBP images of the two versions.
  - When R is increased to 2 and P is increased to 16 (**Figure 27** and **Figure 28**). The most obvious thing is that the LBP image no longer shows the background images that are somewhat similar, but only

retains the seperate lines of the image although it is not clear (This is remind us about Hog's algorithm, which keeps only the main gradient of the image). For the LBP histogram, we notice a rather big change that is, although the shape seems to be similar to the histograms of the previous two versions, the values are now no longer located in the intermediate region and concentrated in the marginal region of the histogram. This represents a restriction on representation of intermediate pixels and instead represents background (0) and seperate line (255). We can see that by increasing the consideration circle's radius R and increasing the number of neighboring points P, we will extract broader features, but that means we will show more influence by unnecessary features (*interference*).

Next, **let's see the results of implementing the LBP algorithm for color images.** The work consists of dividing a color image into 3 RGB channels, then treating each channel as a grayscale image and then implementing LBP algorithm like before. The following results are obtained when implementing the LBP algorithm with $P = 8$ and $R = 1$.

Figure 29: *The resized color image*

Figure 30: *The LBP image of Red Channel*

Figure 31: *The LBP image of Green Channel*

Figure 32: *The LBP image of Blue Channel*

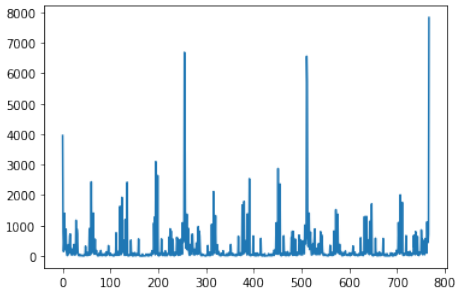

Figure 33: *The combined histogram of 3 Channels*

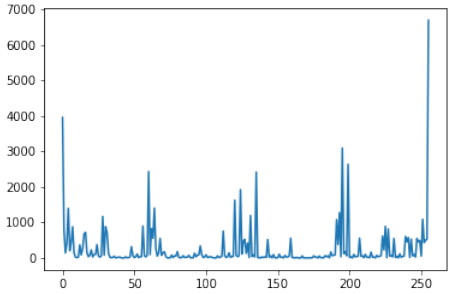

Figure 34: *The LBP histogram of Red Channel*

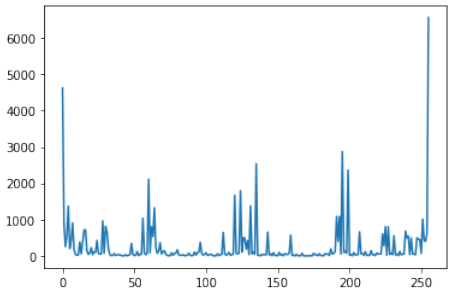

Figure 35: *The LBP histogram of Green Channel*

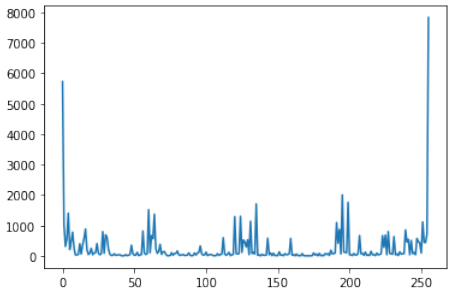

Figure 36: *The LBP histogram of Blue Channel*

Overall, the LBP image or the LBP histogram of the 3 RGB channels of the same color image is not much different, only have different value in a few bins. Finally, to create better results when its come to training in the classification algorithm, instead of graying a color image then calculate the LBP histogram

of it, we can combine the LBP histograms of the three channels of that image into a new histogram of length 256×3, is the representative of that photo.

In addition, we have also implemented the LBP algorithm in Python language (In $P = 8$, $R = 1$ but the way we calculate the LBP is different). Some results are noted as follows:

	LBP for grayscale	LBP for R channel	LBP for G Channel	LBP for B Channel
Library	0.0184 (s)	0.0275 (s)	0.0162 (s)	0.0146 (s)
Self-build	0.4687 (s)	0.4634 (s)	0.4182 (s)	0.4430 (s)

Table 5: *Table of performance results of self-build LBP function compared to library function*

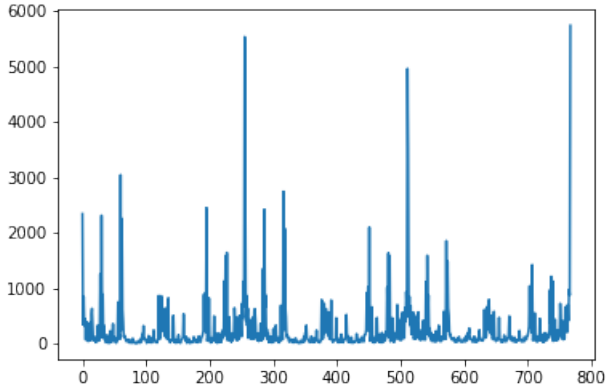

Figure 37: *The self-build combined LBP histogram*

The self-made combined LBP histogram looks quite similar to the combined histogram created by the library but the values in some bins are quite different. This can be explained by doing a different calculation for the LBP value of each pixel, the values of the bins will be swapped compared to the original LBP histogram.

2.1.2 Histogram of Oriented Gradient (HOG)

HOG (Histogram of Oriented Gradients), is a feature descriptor that is often used to extract features from image data. It is widely used in computer vision and image processing tasks for object detection. The concept of HOG was introduced in 1986, however, until 2005 that HOG became widely used after *Navneet Dalal* and *Bill Triggs* published additional documents on HOG. [4]

A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. Typically, a feature descriptor converts an image of size *width* $\times$ *height* $\times$ *3* (channels) to *a feature vector of length n.* In most case of the HOG feature descriptor, the input image is of size $64 \times 128 \times 3$ and the output feature vector is of length 3780. A fact that HOG descriptor can be calculated for other sizes, but in the original paper, the size 64×128 is proposed to use. The essence of the HOG method is to use information about the distribution of *gradient intensities* and *edge directions* to describe local features in the image. Gradients (x and y derivatives) of an image are useful because based on it, we can determine if this is a edge or not and also determine its direction. We have known that edges and corners bring a lot more information about object shape than flat regions and the magnitude of gradients is **large** around edges and corners (regions of abrupt intensity changes), so by calculating the gradients, we easily recognize them. The HOG operators are implemented by subdividing an image into *sub-regions*, called *cells*, and for each cell, we compute a histogram of the directions of the gradients for the points within cell. If we putting the histograms together, then we have a representation of the original image. The histograms are created using the gradients and orientations of the pixel values, hence the name *'Histogram of Oriented Gradients'*. To enhance recognition performance, local histograms can be *contrast-normalized* by calculating an intensity threshold in an area larger than the cell, called *blocks*, and using that threshold value to normalize all cells in the block. The result after the normalization step will be a feature vector that is *more invariant to changes in lighting conditions.*

Let's discuss the step-by-step process to calculate HOG.

Step 1: Preprocess the Data (64×128)

Preprocessing data is *a crucial step* in any machine learning project.

The image need to be preprocessed and brought down the width to height ratio to $1 : 2$. The image size should preferably be 64 x 128 (according to the original paper). This is because we will be dividing the image into 8×8 and 16×16 patches to extract the features. Having the specified size 64×128 will make all our calculations more simple. The paper by *Dalal* and *Triggs* also

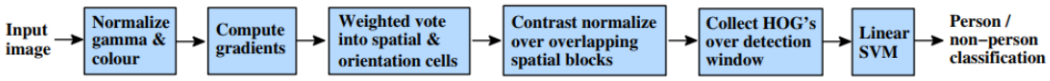

Figure 38: *The overall process of implementing a model to identify a person using HOG (in the original paper)*

mentions **gamma correction** as a pre-processing step, but the performance gains are minor and so just introduce some methods for this (with color images we can do this for all 3 channels of the image) :

- **Gamma rule:** Taking the $\log(p)$ of each pixel value p in the input image .
- **Square-Origin Normalization:** Taking the $\sqrt{p}$ of each pixel p in the input image. By definition, the normal of the square roots compressing the input pixel intensities is *much lower* than the normal standard of gamma.
- **Variance normalization:** We calculate the mean pixel intensity value μ and standard deviation σ of the input image. For each pixel we subtract the mean of the pixel intensities and then normalize by dividing by the standard deviation: $\overline{\mathbf{p}} = (p - \mu)/\sigma$

Step 2: Calculating Gradients (direction x and y)

The next step is to calculate the gradient for every pixel in the image. Gradients are the **change in the x and y directions.** To get the gradient of the image, let's using *convolution*. First, let's define the matrices:

$$\mathbf{D_x} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{33}$$

$$\mathbf{D_y} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{34}$$

Then calculating the gradient by using convolution

$$G_x = \mathbf{I}\mathbf{D_x} \tag{35}$$

and

$$G_y = \mathbf{I}\mathbf{D_y} \tag{36}$$

where $\mathbf{I}$ is input pixel, $\mathbf{D_x}$ is the filter for the x dimension, and $\mathbf{D_y}$ is the filter for the y dimension.

For example, there is a pixel with 4 neighboring pixels in the x and y axis, which are the values to calculate the gradient of that pixel (we don't care much about the value of that pixel when calculating the gradient, but instead we care

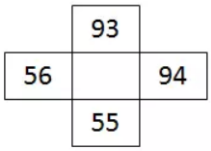

Figure 39: *Example pixel used for calculating gradient*

about the value of neighboring pixels):
Then, use data pixel **Figure 39**, let's start to calculate the gradient of the above pixel.

$$G_x = \mathbf{ID_x} = \begin{bmatrix} 56 & x & 94 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = 38$$

$$G_y = \mathbf{ID_y} = \begin{bmatrix} 93 \\ y \\ 55 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = 38$$

An easier way to understand: To determine the gradient (or change) in the x-direction, subtracting the value on the left from the pixel value on the right. Similarly, to calculate the gradient in the y-direction, subtracting the pixel value below from the pixel value above the selected pixel. Hence apply the formula for that data pixel in **Figure 39** are: Change in X direction:

$$G_x = 94–56 = 38$$

and Change in Y direction:

$$G_y = 93 - 55 = 38$$

The same process is repeated for all the pixels in the image. Then, two new matrices – one storing gradients in the x direction and the other storing gradients in the y direction.
$\longrightarrow$ **The magnitude would be higher when there is a big change in intensity, such as around the edges.** This is why Hog becomes efficient in edge extraction.

Step 3: Calculate the Magnitude and Orientation

Using the gradients calculated in the last step, now let's determine the magnitude and direction for each pixel value.
The **Figure 40** has shown clearly about the way how to calculate gradient and direction. Notice that the notation between G_x and G_y in the **Figure 40** is slightly different from the notation in this part, but they are still the values representing the gradient in the x-direction and the gradient in the y-direction

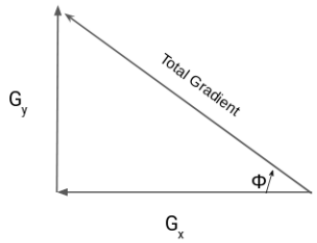

Figure 40: *Calculating the magnitude and direction*

respectively like we mentioned above. For specification, **Total Gradient Magnitude** calculated according to the formula of the theorem:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{37}$$

Next, to calculate the orientation (or direction) for a pixel, let's calculate the angle formed by the gradient of the y and x axis. The value of the angle would be:

$$\phi = \arctan(\frac{G_y}{G_x}) \tag{38}$$

In the *unsigned Hog*, the available range of ϕ is $[0, 180]$

For example, for the pixel in **Figure 39** is:

$$|G| = \sqrt{38^2 + 38^2} = 38\sqrt{2}$$

$$\phi = \arctan\frac{38}{38} = 45$$

Notice that for **color images**, gradients of three channels (RGB) are evaluated. The magnitude of the gradient at a pixel is the maximum value of the gradient intensities of the three channels, and the angle is the angle corresponding to the maximum gradient.

Step 4: Calculate Histogram of Gradients in Cells

Before jump into how histograms are created in the HOG feature descriptor. Let's discuss about some methods to create Histograms using Gradients and Orientation of all pixels in a image.

- **Method 1:** The simplest way to generate histograms is using a histogram of 180 bins (from 0 to 179). Taking each pixel value, find the orientation of the pixel and update the frequency table. We end up with *a frequency table that denotes 180 ranges of angles and the occurrence of these ranges of angles in the image.* This frequency table can be used to generate a histogram with 180 bins of angle values on the x-axis and the frequency on the y-axis.

- **Method 2:** Another method is to create the histogram features for higher bin values. Then, *the number of bins we should use here is 9.* Again, for each pixel in the cell, check the orientation, and calculate the frequency of the orientation values for 9 bins of the histogram, therefore create a 9×1 vector of frequencies.

- **Method 3:** The above two methods use only the orientation values to generate histograms and do not take the gradient magnitude into account. Another way in which can generate the histogram is that instead of using the frequency, *using the gradient magnitude to add the bins in the histogram* (add the gradient magnitude to the bin that the gradient angle fall in).

- **Method 4:** Make a small modification to method 3. In additional, *add the contribution of a pixel's gradient magnitude to the bins on either side of the pixel gradient.* That is, the higher contribution (the higher part of gradient magnitude) should be to the bin value which is closer to the orientation.

As mentioned above, the image is divided into 8×8 cells (the size of the cell can be changed), and the histogram of oriented gradients is computed for each cell. If divide the image into 8×8 cells like above and generate the histograms, we will get a $\mathbf{9 \times 1}$ histogram matrix for each cell. *This matrix is generated using method 4 as we have mentioned* (This is the method that it can represent completely the properties of gradients: using both gradient orientation and gradient magnitude)

Step 5: Normalize gradients in Blocks

Although the HOG feature vectors have been created for the 8×8 cells of the image, *the gradients of the image are sensitive to the overall lighting.* This means that for a particular picture, some portion of the image would be very bright as compared to the other portions. A feature descriptor can't output a result that affected by different intensity levels. They need to be returned to the same reference system in the output (the vector represents the features of an image). Likewise, if there are many different images of the same object with different brightness, it cannot output different vectors of the same object.

The solution is that we can reduce this lighting variation by *normalizing the gradients in a block.* In the case, calculating Hog features of the cells of size 8×8, then use the 16×16 block to normalize the gradient. For specification, combining four 8×8 cells to create a 16×16 block. And we already know that each 88 cell has a 91 matrix for a histogram. So, we would have four 9×1 matrices or a single 36×1 matrix. To normalize this matrix, dividing each of these values by **the square root of the sum of squares of the values (the norm 2 of this** 36×1 **vector)**. $\longrightarrow$ That means we have put the output vectors to *the same reference system of intensity* in which each element of this vector is in $[0, 1]$. So called local histograms can be *contrast-normalized*, as mentioned above.

Step 6: Feature extraction for the complete image

So far, features for blocks of the image have been calculated. Now, let's combine all these to get the features for the final image. In our example, we have the resized image is 64×128 and the size of a block is 16×16. However, the blocks taken from a complete image will not be separated from each other, but instead, they will overlap with each other with the overlapping area will be the area of 2 cells. In our example, there are 105 (7×15) blocks of 16×16. Each of these 105 blocks has a vector of size 36×1 as feature vectors of it. Hence, *the total features for the image would be* $105 \times 36 \times 1 = 3780$ *features in our case, is the all thing we need.*

We also did some experiments with Hog feature extraction with the input image is **LeNa_color.png**. First, let's try with grayscale image:

Figure 41: *The original image*

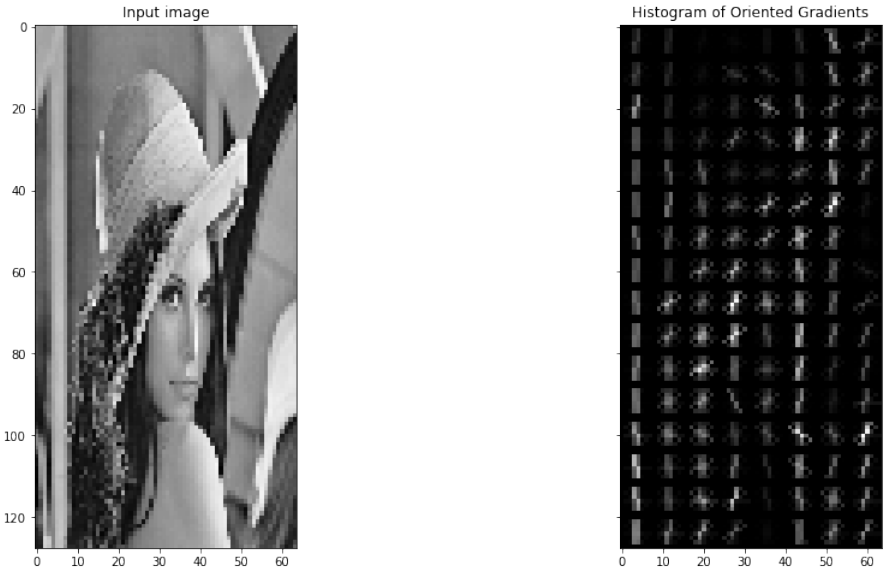

Figure 42: *The resided image(left) and the HOG output image(right)*

We have also obtained a HOG vector of size 3780×1. However, it can be easily seen that the HOG output image is quite unclear because we have specified the intensity of the output vector to be in the range [0,1]. For better display, using the *exposure.rescale_intensity()* function of the library *exposure* to rescale the intensity range of the output image, increase the overall brightness of the photo. Here is the result:

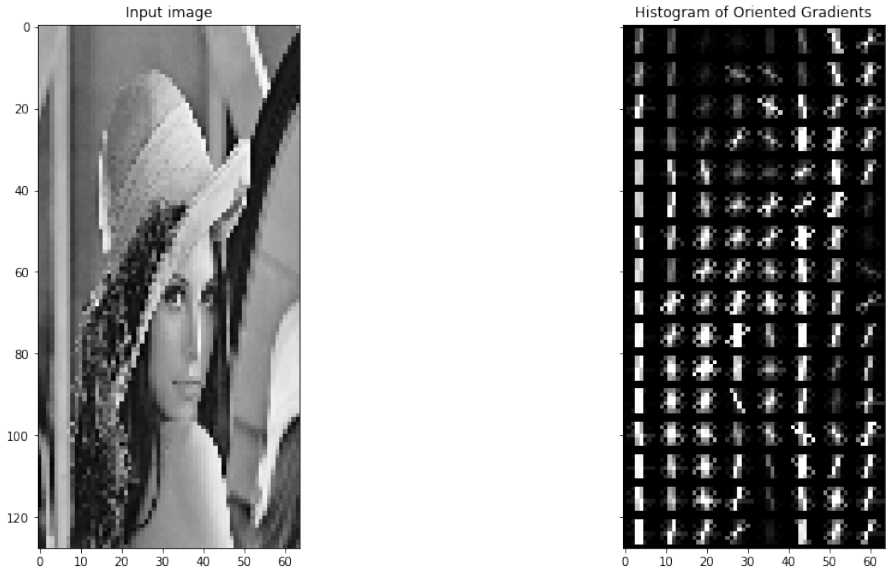

Figure 43: *The resided image(left) and the HOG output image has been rescaled(right)*

Next, let's try the HOG algorithm for color image. Here, we also change some parameters of the algoritm to see what will happen:

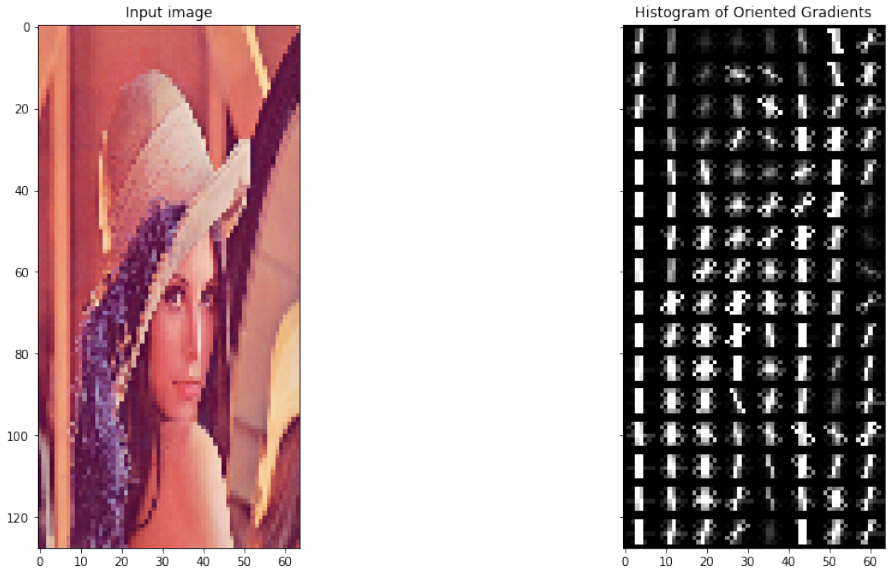

Figure 44: *The color resided image(left) and the HOG output image with* 8×8 *cells and* 64×128 *resized image has been rescaled(right)*

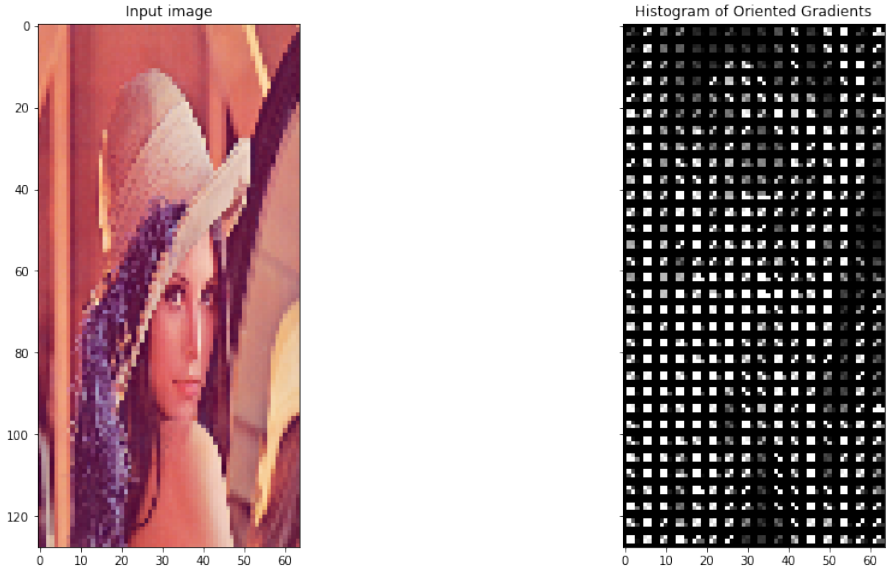

Figure 45: *The color resided image(left) and the HOG output image with* 4×4 *cells and* 64×128 *resized image has been rescaled(right)*

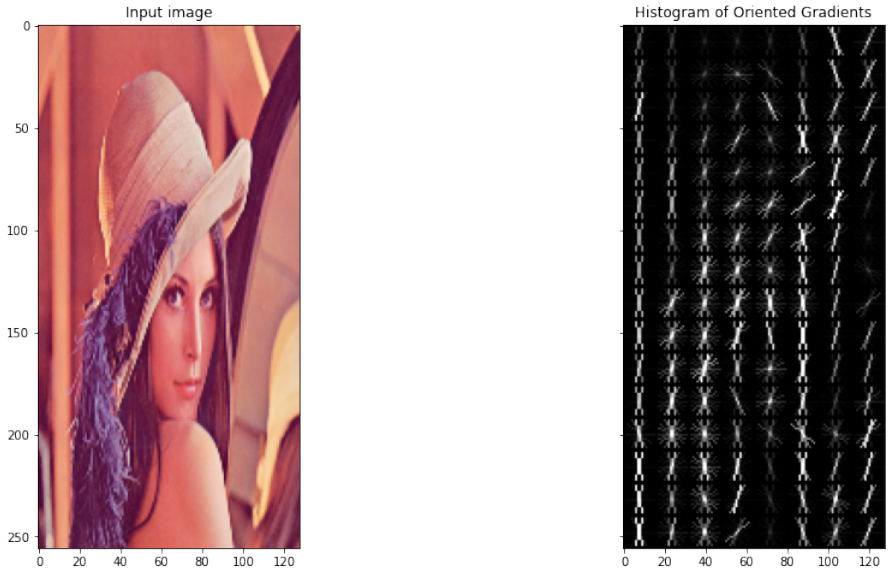

Figure 46: *The color resided image(left) and the HOG output image with* 16×16 *cells and* 128×256 *resized image has been rescaled(right)*

Observation:

- For each cell in the output image, we represent a HOG distribution consisting of *a group of 9 common vectors* whose length is equal to the gradient magnitude and the angle is equal to the gradient direction.
- *Hog encoded an image quite well*, at each pixel the gradient vectors almost completely represented the important edges of an image. Obviously, we can immediately recognize the image of the girl when looking at the output image. Especially. when we reduce the size of the cell to 4×4, we see it even more clearly because we go into smaller areas and represent important features of the image with more vector gradients. However, the price will certainly be that the size of the HOG vector will be much larger.

In addition, we have also implemented the HOG algorithm in Python language (but just in grayscale image and fixed size of cell and input image). Some results are noted as follows:

	HOG image for grayscale 8×8 cells and 64×128 resized image
Library	0.02579 (s)
Self-build	0.14422 (s)

Table 6: *Table of performance results of self-build HOG function compared to library function*

Finally, noticing that the size of the self-build vector is exactly the same as the size of the library-generated vector. After subtracting the two vectors, we see that the mean of the difference vector is about -0.006, which means we have partially implemented the idea from the beginning.

2.2 Text data vectorization

2.2.1 Term Frequency–Inverse Document Frequency (TF-IDF)

TF-IDF is a numerical static method reflects level important of a word in documents. TF-IDF is simple text vectorization used most in text-based recommender systems. [2] Mathematics of TF-IDF TF-IDF includes 2 values: TF-Term Frequency is frequency of occurrence of a word in a document and IDF-Inverse Document Frequence is metric which considers the importance of a word. TF-IDF value of a word is multiply of these values.Suppose that we have corpus $\mathbf{D}$ includes n documents, and $\mathbf{d}$ is a document in the corpus. TF value is calculate as following formula:

$$TF(t, \mathbf{d}) = \frac{f(t, \mathbf{d})}{n(\mathbf{d})} \tag{39}$$

Where:
- TF(t,$\mathbf{d}$) is TF value of word t in document $\mathbf{d}$.
- f(t,$\mathbf{d}$) is frequency of occurrence of word t in $\mathbf{d}$
- n($\mathbf{d}$) number of words in document $\mathbf{d}$.

There are many stop words such as “is”, “of”, “that”, etc in documents have large number of occurrences but they are not important. So IDF value to reduce the value of common words. Each word has only 1 unique IDF value in the corpus.

$$IDF(t, \mathbf{D}) = log\frac{|\mathbf{D}|}{|\{\mathbf{d} \in \mathbf{D} : t \in \mathbf{d}\}|} \tag{40}$$

Where:
- IDF(t,$\mathbf{D}$) is IDF value of word t in corpus $\mathbf{D}$.
- $|\mathbf{D}|$ is number of documnets in corpus.
- $|\{\mathbf{d} \in \mathbf{D} : t \in \mathbf{d}\}|$ is number of documents in $\mathbf{D}$ have word t.

$$TF - IDF(t, \mathbf{d}, \mathbf{D}) = TF(t, \mathbf{d}) \times IDF(t, \mathbf{D}) \tag{41}$$

Implement TF-IDF Step 1: With input is 2 simple short articles, call textA, textB in Vietnamese.

textA	"cục hàng không việt nam vừa đề xuất bổ sung quy hoạch..."
textB	"trước diễn biến phức tạp của dịch covid theo bộ gd ..."

Table 7: *Example articles for TF-IDF*

Next, preprocessing by deleting punctuations, numbers, special symbols.Then make a wordset includes all words of textA and textB.

WordSet = {'đặc', 'chống', 'ốc', 'đích', 'huống', 'nhiễm', 'xã',...}
Step 2: Correspondingly, make 2 ditionaries with number repeat times of which words in textA, textB and put them as vectors follow order of words in WordSet.

	đặc	chống	ốc	đích	huống	nhiễm	xã	...
vecA	0	0	0	2	0	0	4	...
vecB	14	16	2	0	2	4	0	...

Table 8: *Frequency of words in the articles*

Step 3: Compute TF value, IDF value of each word, and then multiply these values to get TF-IDF value, then put in order to make TF-IDF vector.

	đặc	chống	ốc	đích	huống	nhiễm	xã	...
tfidfTextA	0.00000	0.00000	0.00000	0.00149	0.00000	0.00000	0.00299	...
tfidfTextB	0.00260	0.00297	0.00037	0.00000	0.00037	0.00074	0.00000	...

Table 9: *Vectors TF-IDF*

Summary TF-IDF is simple method in text features extraction and efficient in filtering stop words. But it also has disadvantage points, such as the size of output vector of a document the same with number of words in corpus, or the vector has many zeros values, it will be consumes memory and slows down processing. And solution is using sparse matrix to have a good result in processing data.

2.2.2 Word2Vec

Word2vec is a group of related models use to generate word embedding, maping a word in a vector space. The models are two-layer shallow neural networks are trained to simulate the context, and the meaningful association between words in the corpus. Input of Word2vec is a large corpus , after train the neural network, output is a vector space with a word is assigned to a vector. Word2vec has 2 main models are CBOW and Skip-gram.[1] Structure of Word2vec neural networks Both CBOW and Skip-gram has neural network includes 2 layers. Figure 8 describes the structure of Skip-gram model, which model will predict word surrounding while provide a particular word. Contrary to Skip-gram, CBOW will predict center word when provide many words surrounding. Here, we're dive to find out Skip-gram model.

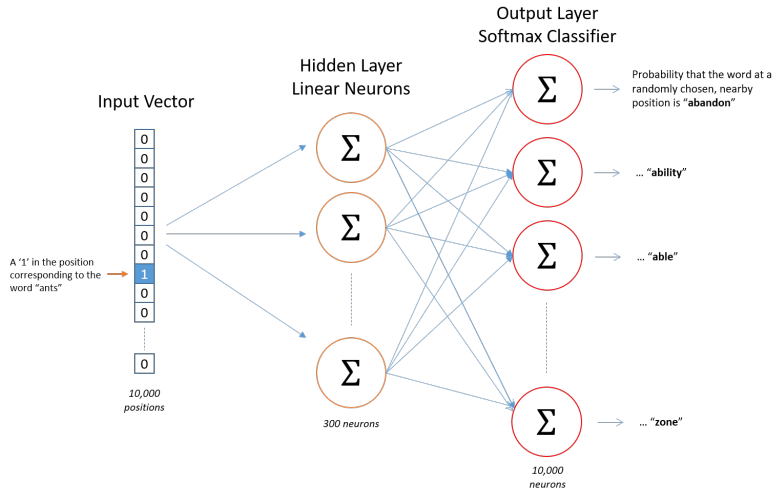

Figure 47: Skip-gram neural network model

Input of the neural network is one-hot vector with same length as number of words in corpus, which encodes by all words in a large corpus, one element is 1 at position of the word in corpus, the rest elements are 0s. Hidden layer usually includes 300 neurons, connect with input layer by weight matrix. Output of the hidden layer is a vector has dimension 300, corresponding representation for the value of the input word.

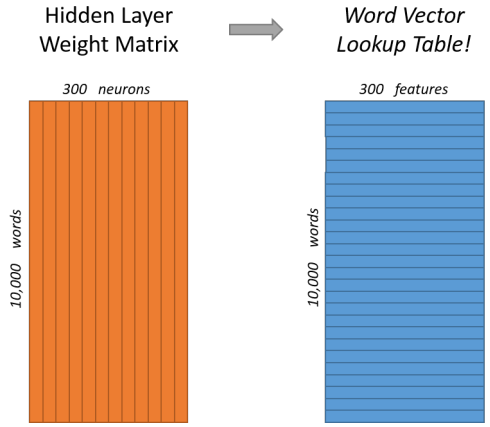

Figure 48: Visualize weight matrix to word vector

The goal of this training the network is to learn from corpus the hidden layer weigth-matrix. Furthermore, how from one-hot vector input of a word can make

a vector size 300 performs for that word.This means that the hidden layer of this model is really just operating as a lookup table. The output of the hidden layer is just the "word vector" for the input word.

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Figure 49: A small example of extract vector from hidden layer matrix

And output layer has the same size as input layer, use softmax activation function to give the probability a random word whether nearby position of the word.

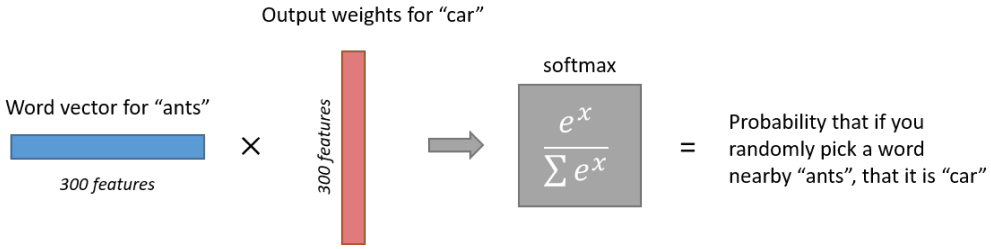

Figure 50: A small example of extract vector from hidden layer matrix

For easy to visualization, figure 9 is an illustration of calculating the output of the output neuron for the word "car". If 2 words have the same or nearby context, through softmax function, should be given a high probability. It means that 2 vectors of 2 words are quite close together in the vector space.
→ Word2vec relies on training a neural network with a hidden layer that performs the representation of words under their circumstances. Thereby, it is possible to vectorize alphanumeric data into numerical vectors that computers can understand and put into machine learning models easily.

References

[1] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *ICLR*. 2013.

[2] Shahzad Qaiser and Ramsha Ali. "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents". In: *International Journal of Computer Applications* 181 (July 2018). DOI: 10.5120/ijca2018917395.

[3] Adrian Rosebrock. "Local Binary Patterns with Python OpenCV". In: (December 7, 2015). URL: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/.

[4] Aishwarya Singh. "Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor". In: (SEPTEMBER 4, 2019). URL: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/.

[5] Philipp Wagner. "Local Binary Patterns". In: (November 08, 2011). URL: https://www.bytefish.de/blog/local_binary_patterns.html.

3 Classification on image, text dataset with Support Vector Machine (SVM)

3.1 Support Vector Machine (SVM)

3.1.1 Mathematics of SVM

In multidimensional space, distance from a point (vetor) to a hyperplane has equation: $\mathbf{w}^T\mathbf{x} + b = 0$ is defined:

$$\frac{\left|\mathbf{w}^T\mathbf{x}_0 + b\right|}{\|\mathbf{w}\|_2} \tag{42}$$

where $\|\mathbf{w}\|_2$ is norm 2 of $\mathbf{w}$.[1]
Let consider two groups of points classifier by find a boundary between them. For easy visualization, we go to learn on two-dimensional space, suppose there are two groups of points as shown and we need to find the dividing line so that the distance from the nearest points to the line is largest and equal. Suppose that pair points of train set is $(\mathbf{x}_1, y1), (\mathbf{x}_2, y2), ..., (\mathbf{x}_n, yn)$ where $\mathbf{x}_i \in R^2$ and y_i is label of this point, 1(blue) or -1(red).

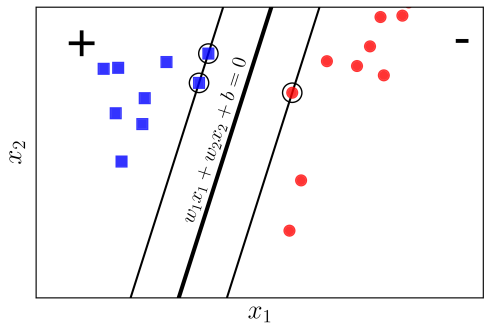

Figure 51: A small example of extract vector from hidden layer matrix

We can defined margin as distance from any point to the boundary:

$$margin = \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_0 + b)}{\|\mathbf{w}\|_2} \tag{43}$$

We have to find w and b such that the margin is the largest:

$$(\mathbf{w}, b) = \underset{\mathbf{w},b}{argmax} \left\{ \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_0 + b)}{\|\mathbf{w}\|_2} \right\} \tag{44}$$

Easy to see that if we replace $\mathbf{w}$ by $k\mathbf{w}$, b by kb then no effect on our result. So we suppose $y_n(\mathbf{w}^T\mathbf{x}_0 + b) = 1$. We got result as figure 2, so with any point, we have $y_n(\mathbf{w}^T\mathbf{x}_0 + b) \geq 1$.

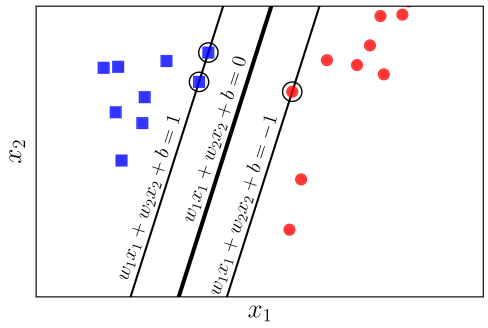

Figure 52: A small example of extract vector from hidden layer matrix

We can rewrite the problem as the optimization problem has the following constraints:

$$(\mathbf{w}, b) = \underset{\mathbf{w},b}{argmin} \frac{1}{2} \|\mathbf{w}\|_2 \, st : 1 - y_n(\mathbf{w}^T \mathbf{x}_0 + b) \leq 1 \tag{45}$$

At this point, this problem can be solved with the formula for the Quadratic Programming convex optimization problem. But when the number of data dimensions is large, things become more difficult, an effective way is to use the Lagrangian dual problem to solve.
Lagrangian of SVM:

$$\mathcal{L}(\mathbf{w}, b, \lambda) = \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{n=1}^{N} \lambda_n (1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)) \tag{46}$$

where $\lambda = [\lambda_1, \lambda_2, ..., \lambda_N]^T$, $\lambda_n \geq 0, = 1, 2, ..., N$
The Lagrangian dual function is defined as:

$$g(\lambda) = \underset{\mathbf{w},b}{min} \mathcal{L}(\mathbf{w}, b, \lambda) \tag{47}$$

where $\lambda \succeq 0$
To find the minimum of this function,solving the system of zero derivative equations of two component variables.

$$\frac{\mathcal{L}(\mathbf{w}, b, \lambda)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n, \frac{\mathcal{L}(\mathbf{w}, b, \lambda)}{\partial b} = -\sum_{n=1}^{N} \lambda_n y_n = 0 \tag{48}$$

Take shorten we get:

$$g(\lambda) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \tag{49}$$

Assigned:

$$\mathbf{V} = [y_1 \mathbf{x}_1, y_2 \mathbf{x}_2, ..., y_n \mathbf{x}_n,] \tag{50}$$

$$\mathbf{1} = [1, 1, ..., 1]^T \tag{51}$$

$$\mathbf{K} = \mathbf{V}^T \mathbf{V} so \mathbf{K} \succeq 0 \tag{52}$$

So $g(\lambda) = -\frac{1}{2} \lambda^T \mathbf{K} \lambda + \mathbf{1}^T$ is a concave function.
Lagrange duality problem:

$$\lambda = \underset{\lambda}{argmax} g(\lambda) st : \lambda \succeq 0, \sum_{n=1}^{N} \lambda_n y_n = 0 \tag{53}$$

This problem can be solved by optimizing Quadratic Programming.[3]

3.1.2 Versions of SVM

+ Soft Margin SVM:
SVM is only work on separately dataset, it means the boundary must completely separate the two sets of points so that no point is misclassified. But in reality, our data can be very complicated, maybe nearly linearly separable, so SVM can't work in this situation. But there is still a way that we will accept the loss at some point will be misclassified.
+ Kernel SVM:
In the situation that our data is completely linearly indistinguishable, then we can use kernel SVM to classify the points, with the basic idea of finding a transformation such that from the original space, the data are linearly indistinguishable to a new space of two linearly separable datasets. And finally, we can use SVM or Soft Margin SVM to solve the problem.

3.2 Images classification with SVM

Image classification is the process of categorizing and labeling groups of pixels or vectors from an image based on some rules (some algorithms). The categorization law can use one or more spectral or textural characteristics. Two general methods of classification are *supervised* and *unsupervised*.

Unsupervised classification method is a automated process without the use of training data. Instead, using a suitable algorithm and the specified characteristics of an image is detected during the image processing stage. The popular classification methods are *clustering* or *pattern recognition*. Two well-known algorithms are *ISODATA* and *K-mean*. In here, **K-mean clustering algorithm** will be used for image classification. The simplest idea of *a cluster* is a collection of points that are close together in some space. Assume each cluster has *a center point* and the points around each center belong to the same group as that center. In the simplest way to understand, considering any point, we consider which point is closest to the center, then it belongs to the same group as that center. The optimization problem here is that when given us a finite number of data points and we are trying to classify it into **k** groups, the distance from a point to the center of that point's group must be the minimized. [4] We can summarize this algorithm as **pseudocodes**:

Input: Data and the number of clusters **K**

Output: Centers for each cluster and label for each data point for minimize the cost function.

- Choose **K** random point as the initial center.
- Assign each data point to the cluster whose center is closest to it.
- If the assignment of data to each cluster in step 2 does not change compared to the previous loop, then we stop the algorithm.
- Update the center for each cluster by averaging all the data points assigned to that cluster after the step 2.

- Go back to step 2.

There guarantees that the algorithm will stop after a finite number of iterations. Indeed, since the lost function is a positive number, and after every step 2 or 3, the value of the loss function is decremented (we can proved this). According to the knowledge of the sequence : if a sequence is decreasing and lower-bounded then it converges. Moreover, the number of ways to group the entire data is finite, so at some point the algorithm will stop. Then we have the final solution for this problem. K-mean Clustering algorithm can be considered as an easily-understanding and simple algorithm, but it has some limitations such as the number of clusters to cluster need to be known, the final solution depends on the centers that are initialized in the beginning, Clusters need to have roughly equal number of points, Clusters need to be circular,...

Supervised classification method is the process of dividing the dataset into train images and input images, with the train images each has a label for the learning process. Before entering the classifier, the images go through several pre-processing methods, feature extraction, or normalization to get the most important features and to increase performance. The **Figure 53** will show the process for supervised image classification. Two well-known algorithms in supervised learning are K-nearest neighbors (KNN) and Support Vector Machine (SVM). SVM has been mentioned quite specifically above, and here we will talk about KNN. **K-nearest neighbor** is one of the simplest supervised-learning algorithms in Machine Learning. When training, this algorithm does not learn anything from the training data (this is also the reason this algorithm is classified as lazy learning), all calculations are performed when it needs to predict the outcome of new data. With KNN, in the Classification problem, the label of a new data point is *directly inferred from the nearest K data points in the training set*. The label of a test data can be decided by *major voting between the closest points*, or it can be inferred by *assigning a different weight to each of the nearest points and then deducing the label*. We can measure the distance between points with norm 1, norm2,... and increase or decrease the number of neighboring points considered to get better results in different cases. The **advantage** of KNN can be easily deduced that its computational complexity of the training process is 0 since we don't need to train anything, the way we predict the outcome of new data is simple and there is no need to make any assumptions about the distribution of classes,... However, KNN is very sensitive to outlier when K is small. Besides that, calculating the distance from the target point to each data point in the training set will take a lot of time, and storing all data in memory also affects the performance of KNN. [2]

Entering the experimental part. First, let's talk about the dataset, which is a small dataset of 130 face images of 13 people of class AI1502. These images are cropped to take only the face (*assuming we skip the face detection step in the face detection problem*) and there was a label corresponding to each image is the name of the person in that photo. Thus, there are **130 input images and 13 classes** to identify. At the same time, using the first 8 images of each class to serve the training process and the remaining 2 images to evaluate the

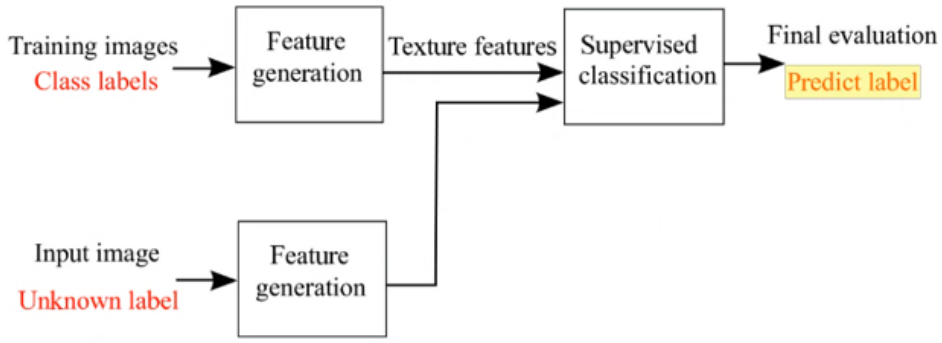

Figure 53: *Procedure for a supervised image classification*

results (in the supervised problem). During the experiment, different methods for feature extraction have been worked with , tested various image classifications (supervised and unsupervised) and also their variations. Notice that we use metrics **Accuracy** and **F1-score** to evaluate the performance of the model.

The first feature extraction method that can be mentioned is **the LBP (Local Binary Pattern) method for the grayscale image**. The input image is read into a grayscale image, then the LBP algorithm is used to calculate the feature vector representing that image. Finally, putting a set of 104 vectors representing 104 input images with their labels into the supervised classification and then using the remaining 26 vectors to predict the output and evaluate the performance.

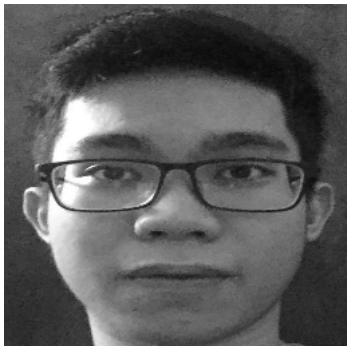

Figure 54: *The grayscale resized image*

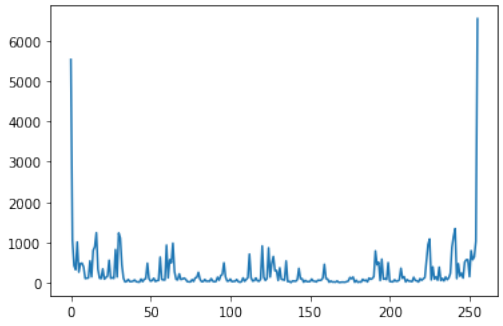

Figure 55: *The LBP output vector with* $P = 8$ *and* $R = 2$

And here are some results after doing some experiments with different image classifiers. First with the SVM classifier:

	LBP with $P = 8$ and $R = 14$	LBP with $P = 84$ and $R = 2$	LBP with $P = 8$ and $R = 2$ with blurring before extraction	LBP with $P = 8$ and $R = 3$	LBP with $P = 16$ and $R = 1$	LBP with $P = 16$ and $R = 1$ with blurring before extraction	LBP with $P = 16$ and $R = 2$	LBP with $P = 8$ and $R = 3$
Accuracy on the test set (%)	73.08	**76.92**	69.23	73.08	76.92	69.23	73.08	73.08
F1-score on the test set (%)	73.08	**76.92**	69.23	73.08	76.92	69.23	73.08	73.08

Table 10: *Table of some results with different parameters in LBP descriptor for grayscale image using SVM*

Observation:

- When using F1-score with the parameter *average = "micro"*, it means we calculate metrics globally by counting the total true positives, false negatives, and false positives and we found that F1-score is always equal to metric *accuracy*. This can be reasoned by f1-score is valid when predicting many objects in the same image (i.e. TP, FP, TN, FN values have a lot of variation). However, here we only predict 1 object in the image. If the image is predicted correctly, then TP++ in the image's class, TN++ with the remaining classes, and if the prediction is wrong, then FP++ for that class, FN++ for the predicted class (but wrong), and TN ++ for the rest of the classes. Given these rules, perhaps calculating f1-score is still the same as calculating accuracy. Therefore in the following examples, we will not present the f1-score value anymore.

- From the above table, it can be observed that with $P = 8$, $R = 2$, and $P = 16$, $R = 1$, the LBP extraction produces the highest accuracy of about **77%**. It is reasonable to say that maybe the input data is suitable with the above two variants of LBP extraction, and the results in the following sections will prove this.

- In the previous task, we thought that blurring the image before entering the feature extraction could give a better result. However, in this case, blurring the image severely reduced accuracy. Presumably, that blur is not necessary for this dataset.

Get the results of the dataset after performing the LBP algorithm with $P = 8$ and $R = 2$ (We choose this parameter because it bring highest result in SVM model) and put it into the KNN classifier, we have the following results:

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights	KNN with neighbors=5, p=1
Accuracy on the test set (%)	76.92	**80.77**	76.92	**80.77**	76.92	76.92	76.92	69.23

Table 11: *Table of some results with different parameters in KNN for applying LBP descriptor to grayscale image*

Observation:

- Using norm 1 achieves higher accuracy than norm 2
- When increasing the number of neighboring points to consider, the accuracy will increase, reach the maximum and then quickly decrease because of outliers.
- Using "distance" weights to add bias to nearby points doesn't work well in this case.

About using *K-mean clustering* for unsupervised classification of input images. First calculating all the LBP vectors of the input images (in here we use P = 8 and R = 2) and then use the K-mean clustering algorithm to classify the images. For every 10 images of the same person, we will use the cluster that the algorithm classifies with the highest frequency as the output class of the algorithm. And this is the result:

Cluster	4	5 and 0	2	1	4	7	0 and 6	4	6	9	2	12 and 7	4
Frequency	3	4	**9**	**10**	**10**	**10**	3	4	**10**	5	**8**	2	**10**

Table 12: *Table of some results in K-mean clustering for applying LBP descriptor to grayscale image*

From the above table, it can be observed that unsupervised clustering gives **bad results**. Only about 5 classes are predicted with a frequency greater than or equal to **90%**. Many classes cannot be classified into a specific class but are classified into many different classes. In addition, there are images of 2 different people but are classified in the same class. This can be explained by the fact that although there are the images of different people, their vectors is quite similar and located close to each other in hyperspace, so they will be assigned to the same group.

The next feature extraction method that we approach is still the LBP algorithm, but instead of using the input as a gray image, we use a color image, *extract the LBP output vector on its 3 channels and then concatenate 3 vectors.* In the end, there will be a vector of length 256×3.

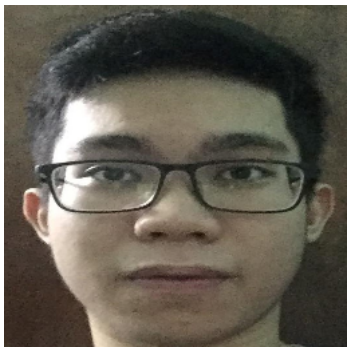

Figure 56: *The color resized image*

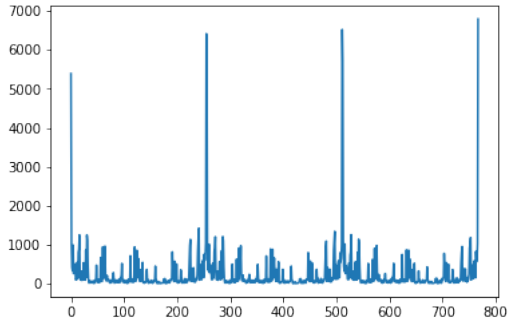

Figure 57: *The LBP output vector concatenated from 3 channels with $P = 8$ and $R = 2$*

And here are some results after doing some experiments with different image classifiers. As usual, starting with the SVM classifier:

	LBP with $P = 8$ and $R = 1$	LBP with $P = 8$ and $R = 2$	LBP with $P = 8$ and $R = 3$	LBP with $P = 16$ and $R = 1$	LBP with $P = 16$ and $R = 2$	LBP with $P = 8$ and $R = 3$
Accuracy on the test set (%)	73.08	**84.62**	80.77	76.92	65.38	69.02

Table 13: *Table of some results with different parameters in LBP descriptor for color image using SVM*

Observation:

- There has been an improvement in accuracy and the highest accuracy we can get is about **85%**. It proves that there can be a better extraction of features when combining 3 LBP vectors of 3 color channels, thereby giving better results.
- LBPs with $P = 8$,$R = 2$, $P = 8$,$R = 3$ and $P = 16$,$R = 1$ are still proving to be good feature extraction methods for this dataset.
- Using "distance" weights to add bias to nearby points doesn't help to increase the accuracy in this case.

Next, let's come to KNN with all the LBP vectors calculated with P = 8 and R = 2 (We choose this parameter because it bring highest result in SVM model) :

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights	KNN with neighbors=5, p=1
Accuracy on the test set (%)	**80.77**	**80.77**	**80.77**	73.08	76.92	73.08	73.08	73.08

Table 14: *Table of some results in KNN for applying LBP descriptor to color image*

Observation:

- The results remain the same, maybe this method is still not strong enough as the extracted features are still quite similar leading to the wrong classifier.
- Using norm 1 achieves higher accuracy than norm 2

When moving to K-mean clustering, there are the following results:

Cluster	12	6 and 3	4	5	0	7	6 and 3	9	4	1	4	9	12
Frequency	3	4	**9**	**10**	**10**	**10**	3	7	**10**	5	**9**	5	**10**

Table 15: *Table of some results in K-mean clustering for applying LBP descriptor to color image*

Observation:

- The results are improved when the cluster classified had a higher frequency than using the LBP vector for grayscale. That means this method makes better feature extraction lead to better classification and the model found more similarities between images of the same person.
- It has not yet shown the diversity in clusters classified when the number of classes classified with high frequency is still very small (about 5 of 13 classes).

Next, let's move to **HOG (Histogram of Oriented Gradients)** feature extraction. The theory of the HOG algorithm has been well explained in the previous section and here we will dive into the practical part. For specification, each image in the dataset will be calculated the HOG vector before feeding into the image classifier.

And here are some results we tested with SVM:

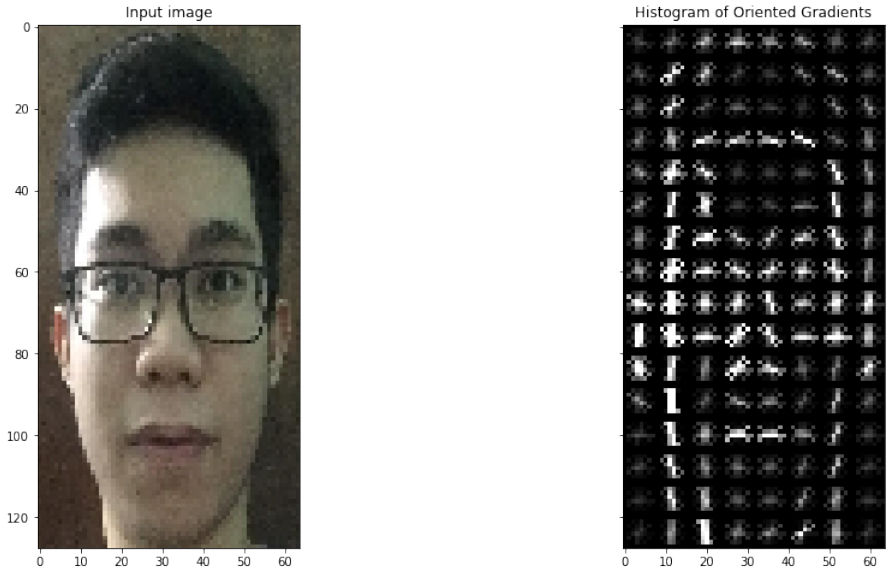

Figure 58: *The color resided image(left) and the HOG output image with* 8×8 *cells and* 64×128 *resized image has been rescaled(right)*

	HOG feature descriptor with image size 64×128 and cell size of 8×8	HOG feature descriptor with image size 64×128 and cell size of 4×4	HOG feature descriptor with image size 128×256 and cell size of 16×16	HOG feature descriptor with image size 128×256 and cell size of 8×8
Accuracy on the test set (%)	**88.46**	**88.46**	**88.46**	**88.46**

Table 16: *Table of some results with different parameters in HOG descriptor for color image using SVM*

Observation:

- Using HOG algorithm in feature extraction gives us *a good and stable result*. Specifically, all 4 variants of hog give us an accuracy of about **88%**. This can be explained by that the HOG algorithm has *extracted features on small regions of the image and is represented individually* by parts on the HOG vector (in contrast to LBP, features will be computed over the whole image and merged into a vector of length 256). This has increased the accuracy of the model, but the length of the feature vector for the hog algorithm is much longer (With the resized image size is 64×128 and the cell size is $8 \times$)
- Feature extraction of small parts in the image (we call it in the *blocks*) may increases the computational load and reduces the overall performance

of the model. However, this problem can be mitigated by the data dimensionality reduction method (specifically **PCA**) which will be discussed in the next section.

Coming to the KNN algorithm, we have the following positive results (all with HOG feature descriptor with image size 64×128 and cell size of 8×8):

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights	KNN with neighbors=5, p=1
Accuracy on the test set (%)	88.46	88.46	88.46	**92.31**	88.46	88.46	88.46	84.62

Table 17: *Table of some results in K-mean clustering for applying HOG descriptor to color image*

Observation:

- The results are quite *stable and even*. The highest accuracy is about **92%**, when we use the number of neighbors is 3 and norm 1.
- When using norm 2 instead of norm 1, the result is also considered stable however the performance is somewhat lower.

Finally, let's come to the experiment with K-mean clustering, and here are the results:

Cluster	0	7	11	2	1	12	9	8	5	4	4	9	3
Frequency	7	**10**	**8**	**10**	**10**	**8**	6	7	**10**	**8**	**9**	4	**10**

Table 18: *Table of some results in K-mean clustering for applying HOG descriptor to color image*

Observation:

- It can be seen that the results are *much better* when the predicted clusters are somewhat *more diverse and at a much higher frequency* than previous tests with LBP. Specifically, up to 10 different clusters are predicted for 10 groups of photos of 10 people with high frequency (greater than or equal to 70% frequency and 5 cluster groups are predicted with perfect 100% frequency with no repeats in other groups of images).

- *The efficiency of feature extraction in small blocks* has been demonstrated when even though there are some identical predictions. That means that feature vectors of photos were more differentiated between different groups and there was a sharper similarity in the same group of photos (photos of the same person).

Another idea that we've been thinking about is that using **a combination of the LBP and HOG algorithms**. With LBP removing the backgrounds leaving only the important edges of an image, then using HOG algorithm on that LBP image in the hope of extracting the gradient vector of those edges efficiently and with less affected by noise.

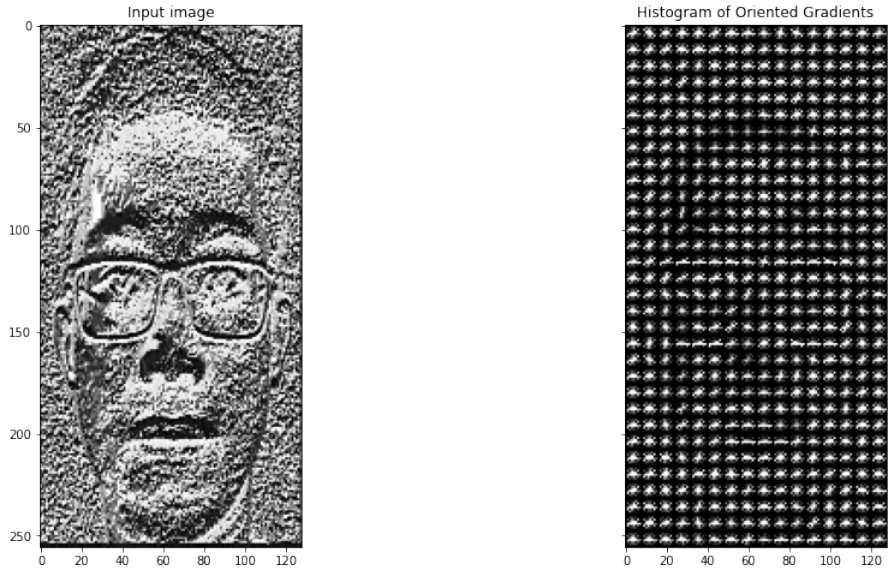

Figure 59: *The resized image with size 128 × 256 after applying LBP algorithm with $P = 8$, $R = 2$(left) and The HOG output vector calculated with cell size 8×8 in that LBP image(right)*

And here are some results when using the SVM image classifier:

	LBP($P = 8$, $R = 1$) for grayscale image then take HOG with resized image size 64×128 and cell size 8×8	LBP($P = 8$, $R = 2$) for grayscale image then take HOG with resized image size 64×128 and cell size 8×8	LBP($P = 8$, $R = 3$) for grayscale image then take HOG with resized image size 64×128 and cell size 8×8	LBP($P = 16$, $R = 1$) for grayscale image then take HOG with resized image size 64×128 and cell size 8×8	LBP($P = 16$, $R = 2$) for grayscale image then take HOG with resized image size 64×128 and cell size 8×8	LBP($P = 16$, $R = 3$) for grayscale image then take HOG with resized image size 64×128 and cell size 8×8	LBP($P = 8$, $R = 2$) for grayscale image then take HOG with resized image size 128×256 and cell size 8×8
Accuracy on the test set (%)	88.46	88.46	92.31	92.31	92.31	88.46	**96.15**

Table 19: *Table of some results with different parameters in LBP-HOG descriptor for grayscale image using SVM*

Observation:

- There has been *a marked improvement in accuracy*, which shows that our approach has worked. In particular, when we use the LBP algorithm with P=8, R=2 then use the HOG algorithm on that LBP image with the resize image size of 128×256 (larger than the original) then use If the cell size is 8×8 (keeping the same cell size), the result is **96%** accurate, the highest we have ever achieved. However, the cost is large when the length of the feature vector is up to 16740, causing a decrease in the performance of the model in computation and storage.

- The length of the feature vector at about 16000 will give the best results, the results of the next experiments will prove it.

Let's come to KNN classifier's result (Here we use LBP($P = 8$, $R = 2$) for grayscale image then take HOG with resized image size 128×256 and cell size 8×8 because it bring the best result) :

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights	KNN with neighbors=5, p=1
Accuracy on the test set (%)	**80.77**	**80.77**	**80.77**	73.08	76.92	73.08	73.08	73.08

Table 20: *Table of some results in KNN for applying LBP descriptor then HOG descriptor to grayscale image*

Observation:

- Overall results seem diminished, although we expect it will be higher. Maybe when combining the two feature extractions together, the images have been stripped of many different components, leading to the data points being located closer together, leading to a lot of incorrect classification.
- Adding a bias based on distance from the center worked, though there was no significant increase in accuracy.
- Testing with norm 2 brings worse results than using norm 1.

When it comes to K-mean clustering, we have the following results with the method using LBP($P = 8$, $R = 2$) for grayscale image then take HOG with resized image size 128×256 and cell size 8×8:

Cluster	3	3	11	2	12	11	3	3 and 11	9	0	0	11	4
Frequency	4	6	5	**10**	**9**	**9**	5	5	**9**	4	**9**	5	**9**

Table 21: *Table of some results in K-mean clustering for applying LBP descriptor then HOG descriptor to grayscale image*

Observation:

- This result is *an acceptable result* when there are 6 different clusters classified with frequency greater than or equal to 90. Looking at the dataset, these are the clusters of quite similar images and quite easy to classified. However, a good point we can see here is that with the more difficult images, there is a *clearer classification* when a group of images of a person is no longer classified into too many different clusters.

Another idea that came to mind was when we looked at the HOG method, which split the image into cells and blocks for extraction. That is, in HOG we

have extracted the feature in the smaller parts of the image and keep it as a part of feature vector, not aggregating like in LBP (In LBP we compute the feature in each pixel of the image then aggregated over the entire image, not kept as part of the vector like HOG). In previous tests, when calculating the feature LBP in the same way, the result was not very high, maybe it was because the vectors were not able to show the features of small areas of the image in the feature vector, making it difficult for the classifier to classify them. So, why don't break down the LBP vectors of an image's components to become part of the feature vector instead of bundling them together. And that led to a new idea when splitting the image into small equal sized parts (we call it *blocks*), calculate the LBP vector of the blocks then concatenate it into a feature vector. Maybe we call it **block-splitted LBP**. First, **let's try the grayscale image**:

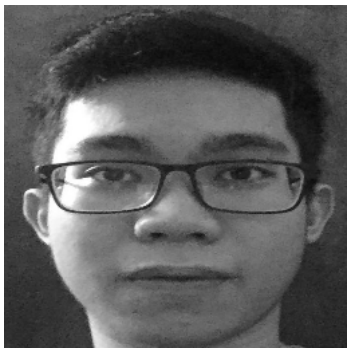

Figure 60: *The grayscale resized image of size 256 × 256*

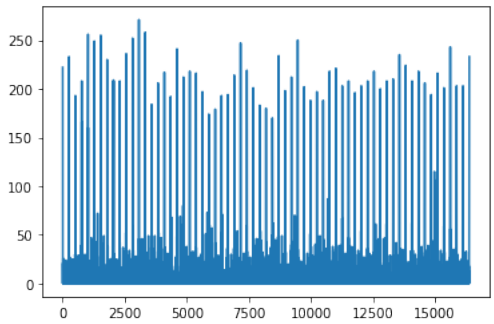

Figure 61: *The LBP output vector with 32 × 32 block size, P = 8 and R = 1*

Let's consider the SVM classifier:

	LBP with $P = 8$ and $R = 1$	LBP with $P = 8$ and $R = 2$	LBP with $P = 8$ and $R = 3$	LBP with $P = 16$ and $R = 1$	LBP with $P = 16$ and $R = 2$	LBP with $P = 8$ and $R = 3$
Accuracy on the test set (%)	92.31	**96.15**	**96.15**	**96.15**	**96.15**	**96.15**

Table 22: *Table of some results with different parameters in blocked LBP descriptor for grayscale image using SVM*

Observation:

- It is *an extremely good result* as 5 out of 6 variants of LBP fed into the SVM classifier gave very high results (about **96%**), almost the entire image was predicted (according to our data, note that the unpredictable image is a face not in the direct direction but in the inclined angle, perhaps

we need an even better feature extractor for absolute results). However, with 96% results in most variants, we can admit that extracting LBP features in small parts of the image and then concatenating them has brought incredible efficiency, proving this method to be *a strong method* and suitable for the dataset we have.

- However, *the price to pay is also expensive* .If there was the resized image of size 256×256 and the block size was 32×32, the LBP vector had a length of 256×64 because that image will have 64 blocks. It will be a big price when the data becomes bigger because it will *take a lot of time for computation and storage*, and in such cases, a data dimensional reduction is absolutely necessary.

When it comes to KNN, LBP algorithm with parameters $P = 16$,$R = 1$ and $P = 8$,$R = 2$ for feature extraction is used because through many past experiments, these are the parameter values that gives the best results. With $P = 16$ and $R = 1$:

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights
Accuracy on the test set (%)	**96.15**	92.31	**96.15**	92.31	**96.15**	92.31	92.31

Table 23: *Table of some results in KNN for applying blocked LBP descriptor to grayscale image*

With P = 8 and R = 2:

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights
Accuracy on the test set (%)	**96.15**	92.31	**96.15**	92.31	**96.15**	92.31	**96.15**

Table 24: *Table of some results in KNN for applying blocked LBP descriptor with different version to grayscale image*

Observation:

- This is an good result as all tests give high results, all above **92%**.

- It proved that separating the features of parts of an image into a separate segment of the feature vector contributed to more clearly seperating images.

In the end, let's come to see the result of K-mean clustering classifier:

Cluster	10	5	1	4	9	6	10	8	1	3	1	2	0
Frequency	5	6	**9**	**10**	**10**	**10**	7	7	**10**	6	8	6	**10**

Table 25: *Table of some results in K-mean Clustering for applying blocked LBP descriptor to grayscale image*

Observation:

- An acceptable result as most of the clustering is associated with high frequency (all above **50%**).
- Although there are some groups of images with the same cluster that are predicted. However, this is an unsupervised algorithm so it can be accepted.

Another variation inspired by LBP is that instead of reading the gray image and then performing the block LBP extraction, we extract the 3 channels of the color image, perform the same extraction on each channel, and then concatenate them with together. However, with this extraction, the LBP feature vector in this case will be three times as long as the feature vector in the last version **(3 $\times$ 256 $\times$ 64)**. This is roughly equivalent to spreading the entire 256 $\times$ 256 resized image into a feature vector. Because the vector has such a large length, the classifiers maybe tend to reduce the performance and accuracy of prediction (It called **Curse of dimensionality**).

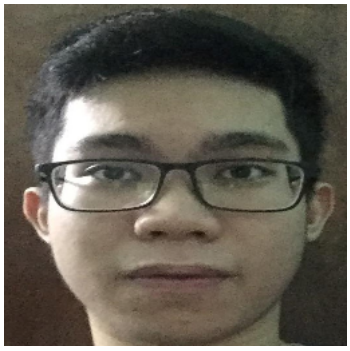

Figure 62: *The color resized image of size 256 $\times$ 256*

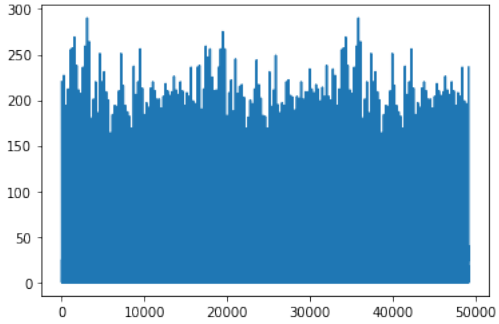

Figure 63: *The LBP output vector with 32 $\times$ 32 block size, $P = 16$ and $R = 1$ has been concatenated*

Here are some results we got with SVM:

	LBP with $P = 8$ and $R = 1$	LBP with $P = 8$ and $R = 2$	LBP with $P = 8$ and $R = 3$	LBP with $P = 16$ and $R = 1$	LBP with $P = 16$ and $R = 2$	LBP with $P = 8$ and $R = 3$
Accuracy on the test set (%)	88.46	**92.31**	**92.31**	**92.31**	**92.31**	**92.31**

Table 26: *Table of some results with different parameters in blocked LBP descriptor for color image using SVM*

And with KNN for LBP output vector with $P = 16$ and $R = 1$ with block size of 32×32:

	KNN with neighbors=1, p=1	KNN with neighbors=2, p=1	KNN with neighbors=2, p=1, using "distance" weights	KNN with neighbors=3, p=1	KNN with neighbors=3, p=1, using "distance" weights	KNN with neighbors=4, p=1	KNN with neighbors=4, p=1, using "distance" weights
Accuracy on the test set (%)	**96.15**	92.31	**96.15**	92.31	**96.15**	92.31	**96.15**

Table 27: *Table of some results in KNN for applying blocked LBP descriptor to color image*

and finally, with K-mean Clustering:

Cluster	12	0	2	1	4	5	12	10	2	3	2	2	7
Frequency	5	**8**	**9**	**10**	**10**	**10**	7	**10**	**10**	6	**8**	6	**10**

Table 28: *Table of some results in K-mean Clustering for applying blocked LBP descriptor to color image*

Observation:

- **With the SVM classifier:** the result, although there is a decrease in accuracy, is still considered to be high (about **92%**). We hoped that as the vector length got larger, the data points would be more distinct from each group and the SVM classifier would give absolute results. However, things are not as we thought because of *curse of dimensionality*.
- **With KNN classifier and K-Mean Clustering:** It is predicted that there would be a reduction in accuracy like the SVM classifier, but the results were really surprising. With **KNN**. the results are not changed compared to the same method with grayscale images (although very high with **96%** for the highest, we had hoped for absolute accuracy). With **K-Mean Clustering**, there was a significant increase in the frequencies of the predicted clusters, specifically, 6 clusters were predicted with **100%** frequency. Although there are duplicates, it is acceptable.

Conclusion:

- The highest accuracy result of the **SVM classifier** is **96.1538%**, this results belonging to many different feature extractions (but they are all computationally intensive and large vector length extractions, so to increase performance input data, a dimensionality reduction is necessary)
- With **KNN classifier** is also similar.
- With **K-mean Clustering**, the best results are obtained when block-based LBP feature extraction on 3 channels of a color image with all frequencies of clusters is high (more than **50%**)
- The longer the feature vector's length increases, the better the results of **KNN** and **K-means Clustering**. With **SVM**, the feature vector length gives the best results is at about **16000** in this dataset.
- **LBP with parameters** $P = 8$**,** $R = 2$ **and** $P = 16$**,** $R = 1$ proved to be suitable for the input data set, always giving good results.
- *Extracting small components of the image and then retaining it as part of the output vector* gives good results in this dataset.
- Feature extraction in 3 channels of color images and then concatenate again compared to feature extraction only in gray images *does not improve the results too much in this dataset.*

In this presentation, we only focus on how to achieve the best results for image classification without much concern about performance. We will cover it in the following section.

3.3 Text classification with SVM

Nowadays, with the development of science and technology, especially in information technology, data is becoming bigger and bigger. Among them, textual data plays an important role in storing and conveying information to people. To easily manage and take advantage of text data, one of the things to do is to categorize them according to the subject matter that the text contains. In this section, we build a small model based on SVM to classify the content of some articles in Vietnamese to consider the factors affecting the results of the model.

3.3.1 Dataset

We used a small corpus that is contributed by members in our class about 12 topics in daily life from some online article websites in Vietnam such as vtcnews.vn, baomoi.com, zingnews.vn,... After preprocessing as deleting punctuations, special signs, and make token by tokenizer, other properties of the corpus is in table 9 below.

Topic	Label	Num of articles	Max length	Min length
Job	VIEC LAM	20	292	93
Culture	VAN HOA	20	543	152
Religion	TON GIAO	20	448	36
Sport	THE THAO	20	1212	213
Health	SUC KHOE	20	608	190
Law	PHAP LUAT	20	228	39
Education	GIAO DUC	20	162	31
Travel	DU LICH	20	213	89
Technology	CONG NGHE	20	258	95
Politic	CHINH TRI	20	186	51
Property	BAT DONG SAN	20	586	96
Music	AM NHAC	20	487	35
Corpus		240	1212	31

Table 29: *Properties of the corpus*

Before putting data into TF-IDF to vectorize text data into numeric vectors, we need to tokenize the data. That means we have to separate documents into meaningful words and phrases (because in Vietnamese sometimes phrases have more common meanings and carry the information they want to convey rather than individual words). We can use *word_tokenize* of library Sklearn to do this task.
Example: "hàng loạt cổ phiếu lớn chìm trong sắc đỏ khiến..."
⇒ "hàng_loạt cổ_phiếu lớn chìm trong sắc đỏ khiến..."

3.3.2 SVM with Sklearn.svm.SVC

Continuously, we use TfidfVectorizer to vectorize documents to numeric vector as sparse matrices. There are some parameters we need to attend, max_df = 0.8 means that words have frequence of occurence in corpus will be removed, these words most likely stop words and don't contribute much to content of documents. max_features = 1000, the function will build a vocabulary that only consider the top 5000 ordered by term frequency across the corpus. So that the size output of data will be (1,1000) for a document. Then split original dataset to train set and test set with the ratio 0.8:0.2, so train set will have 192 articles, and test set will have 48 articles.
Obviously, these vectors are very difficult to be linearly separable, so applying normal SVM or soft margin SVM will bring bad results. So we use the SVM kernel with different kernels as linear, poly, sigmoid and different number of features of the TFIDF vector to look at the results.

n \ *kernel*	linear	poly	sigmoid
50	0.3958, 0.3751	0.4375, 0.4071	0.2916, 0.2624
100	0.6250, 0.6019	0.6875, 0.6598	0.5208, 0.5004
200	0.7916, 0.7737	0.8125, 0.7918	0.7083, 0.6979
300	0.8333, 0.8113	0.8125, 0.7920	0.7708, 0.7527
500	**0.8958, 0.8933**	**0.8958, 0.8920**	**0.8958, 0.8933**
1000	0.8541, 0.8480	0.8750, 0.8708	0.8541, 0.8480

Table 30: *Accuracy and F1-score of kernel SVM versions with different max_feature*

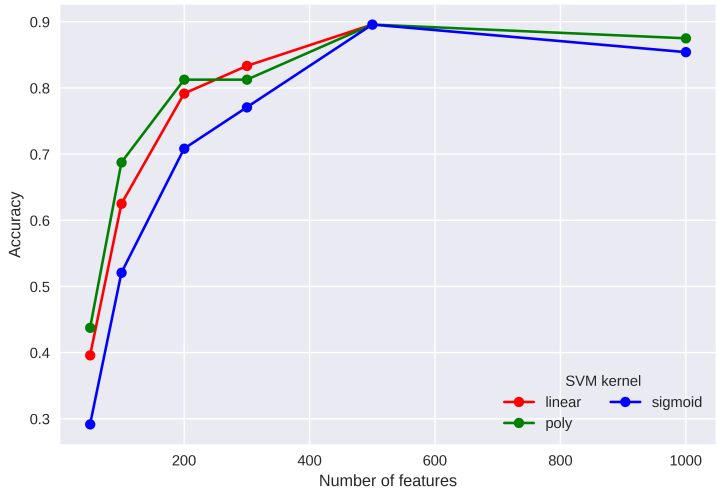

Figure 64: Histogram of median house value

Here, n is max_feature at function TfidfVectorizer. Notice that mean number of words in a documnet is about 213, so n in approximately with the mean with bring best results. With n exceeds 300, accuracy of kernel SVM models have signs of convergence. However, with all kernel, too many features make the model difficultly to find out new vector space that will be linearly separable. So this make accuracy decreases when number of features too high. Because this is small corpus, and all versions of SVM reach best accuracy with vectors has number of dimension is 500. Obviously, the average word count is 213 words, so less than 200 features won't describe all the information for a well-functioning classifier. 500 features carries quite a lot of important information, and 1000 features carries too much information that is not of the article, thus reducing accuracy.

3.4 Data dimensionality reduction with Truncated SVD

With a small Vietnamese corpus, a document as a vector with 1000 features is not good for processing and store. Especially, it is represented as a sparse matrix, so it is necessary to use truncated SVD to reduce the number of data dimensions to 100 for each document. Retains most of the information of the data without much loss of accuracy. Speed up the computation and processing of our model. We can see results in table 11:

Original features	linear	poly	sigmoid
1000	0.8750	**0.9166**	0.8541
500	0.8750	**0.9166**	0.8541
300	0.8541	0.8333	0.8125
200	0.7708	0.8125	0.7708

Table 31: *Results of kernel SVM versions use Truncated SVD from max_features 1000, 500, 300, 200 to max_features = 100*

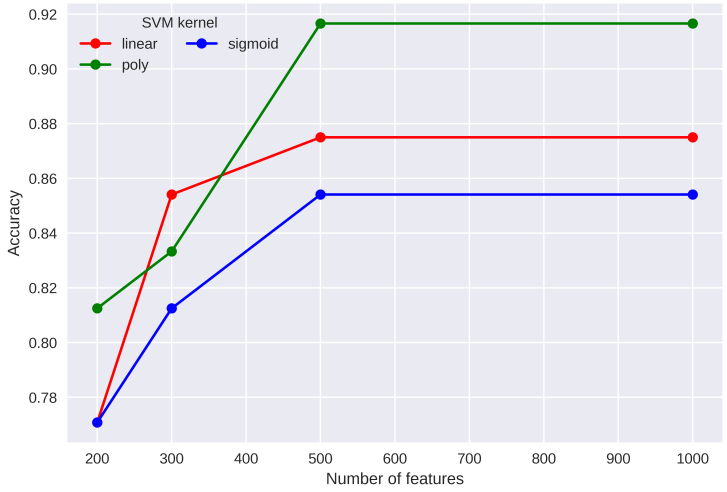

Figure 65: Histogram of median house value

As argued earlier, 1000 features or 500 features when truncated SVD reduced to 100 will both give the same result. because it has removed less important features, but when 500 features or less, accuracy has tended to decrease quite clearly. And there is an interesting point that the accuracy of the poly kernel when using truncated SVD (0.9166) is higher than the original (0.8958). The reason is because when reducing from 500 features to 100, the data loss is not too much. And with the polynomial kernel it is easy to search the new vector space with dimension 100 compared to the original 500, thus giving better results.

3.5 Summary

SVM with different versions is suitable for models in different data cases. Select the version of SVM accordingly can bring best result. Besides, the application of data dimension methods helps increase performance and memory optimization. Notice that parameters affect the amount of information lost. From this simple text classification model, we can see the benefit of topic classification of short articles. When building a large enough corpus, SVM can be applied to classify text content from short to long content on social networking sites, forums or documents. We can build this model completely in Vietnamese to be able to classify more topics.With the current development, especially the text content on social networks is getting larger and larger and providing many valuable sources of information, the development of a text classifier can be the first step to progress towards building systems analysis text information from the above sources.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Jason Brownlee. "Develop k-Nearest Neighbors in Python From Scratch". In: (October 24, 2019). URL: https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/.

[3] Tiep Vu Huu. "Support Vector Machine". In: (Apr 9, 2017). URL: https://machinelearningcoban.com/2017/04/09/smv/.

[4] Shehroz S. Khan and Amir Ahmad. "Cluster center initialization algorithm for K-means clustering". In: *Pattern Recognition Letters* 25.11 (2004), pp. 1293–1302. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2004.04.007. URL: https://www.sciencedirect.com/science/article/pii/S0167865504000996.

4 Principal Component Analysis (PCA)

4.1 Applying PCA for image classification

For a real machine learning problem, the input data often has a very large number of dimensions. If doing calculations and storage on this data space, it will take a lot of computational effort, time and storage space. Therefore, **a data dimensionality reduction** is absolutely necessary for most practical machine learning problems. Data dimensionality reduction, in simple terms, is an algorithm that takes as input a data point in a data domain of dimension N and outputs a new data point in a space of dimension K with $K \ll N$. **Principal Component Analysis (PCA)** is considered as an effective data

dimensionality reduction. *Principal Component Analysis* like other dimensionality reductions is meant to find and retain the most important data. However, if in the worst case, the components of a data point all contain the same amount of information, therefore finding a new way of looking at it (a base transformation of the data space) such that the importance of the data point is clearly seperated. Since the importance of the data point is clearly differentiated, it is easy to leave out the less important components. Specifically, PCA will look for *a new orthogonal system to base the input data space.* The reason this method is called Principal Component Analysis is because the mathematical nature of the method involves *the eigenvalues of the matrix* (called principal components of a matrix). [**PCA** , 1] From a Statistical perspective, PCA can be thought of as a method of finding an orthogonal basis system as using a basis transformation, such that in this new basis of space, the variance in some dimensions is very small, and we can ignore it.

In addition, a different data dimensionality reduction is mentioned that using **Feature Selection** with using the evaluation function to evaluate the importance of each attribute. The evaluation is done by calculating *the correlation* between the attributes to see which attributes are considered dependent on each other (preferred rejected) or independent (preferred retained) and also take care of the class(output) of each data point in the way with that output, which attributes should be used. [3] And here are some results when using data dimensionality reduction on our dataset along with using different feature extraction methods. First, let's consider applying PCA with SVM classifier. Data dimensionality reduction using the PCA algorithm is done through the *PCA()* function in the *sklearn.decomposition package.* The *PCA()* function takes as an input parameter k, where if k is an integer, it is the number of dimensions we want to keep (Note: k must be less than or equal to the rank of the input data matrix) or if k is is a decimal in the range $[0, 1]$ then that is the ratio of the amount of important information we want to keep compared to the total amount of original information,

	LBP extraction for grayscale image	LBP extraction for color image	HOG extraction for color imgae	LBP and HOG combined extraction for grayscale image	Blocked LBP extraction for grayscale image	Blocked LBP Extraction for color image
Highest results from task 3						
Dimension of input space	256	768	3780	16384	49152	16740
Best accuracy on the test set (%)	76.92	84.62	88.46	**96.15**	**92.31**	**96.15**
Running time for classification (s)	0.0075	0.0204	0.1461	0.4388	1.2713	0.5474
Applying PCA with $k = 0.95$						
Dimension of input space	5	6	95	114	54	55
Best accuracy on the test set (%)	61.54	76.92	88.46	**96.15**	**96.15**	**92.31**
Running time (s)	16.2721	15.2148	0.0054	0.0083	0.0050	0.0043
Applying PCA with $k = 90$ features						
Dimension of input space	90	90	90	90	90	90
Best accuracy on the test set (%)	76.92	84.62	88.46	**92.31**	**96.15**	**92.31**
Running time (s)	0.0078	0.0089	0.0062	0.0080	0.0072	0.0043
Applying PCA with $k = 50$ features						
Dimension of input space	50	50	50	50	50	50
Best accuracy on the test set (%)	76.92	84.62	88.46	88.46	**92.31**	**92.31**
Running time (s)	0.0067	0.0064	0.0065	0.0080	0.0043	0.0083
Applying PCA with $k = 20$ features						
Dimension of input space	20	20	20	20	20	20
Best accuracy on the test set (%)	76.92	88.46	88.46	80.77	**92.31**	**92.31**
Running time (s)	0.0075	0.0047	0.0046	0.0027	0.0060	0.0062

Table 31: *Table of results in SVM classsifier after applying PCA*

Next, let's focus on using Feature Selection to reduce the input data dimension. This is done using the *SelectKBest()* function in the *sklearn.feature_selection* package and using the *f_classif* evaluation function also included in that package. The *SelectKBest()* function takes as an input the parameter k which is the number of most important data dimensions we want to keep.

	LBP extraction for grayscale image	LBP extraction for color image	HOG extraction for color imgae	LBP and HOG combined extraction for grayscale image	Blocked LBP extraction for grayscale image	Blocked LBP Extraction for color image
Applying Feature Ranking with $k = 0.95$						
Dimension of input space	230	691	3402	15066	14445	44236
Best accuracy on the test set (%)	76.92	84.61	88.46	**96.15**	**96.15**	**92.31**
Running time (s)	0.0096	0.0173	0.1306	0.5505	0.5769	0.39622
Applying Feature Ranking with $k = 0.70$						
Dimension of input space	179	537	2646	15066	11719	34406
Best accuracy on the test set (%)	73.08	88.46	88.46	**96.15**	**96.15**	**92.31**
Running time (s)	0.0052	0.0164	0.3825	0.5507	0.4536	0.3944
Applying Feature Ranking with $k = 0.50$						
Dimension of input space	128	384	1890	8037	8370	24576
Best accuracy on the test set (%)	84.62	88.46	88.46	**96.15**	**96.15**	**92.31**
Running time (s)	0.0035	0.0164	0.0618	0.2782	0.2534	0.3962
Applying Feature Ranking with $k = 100$ features						
Dimension of input space	100	100	100	100	100	100
Best accuracy on the test set (%)	73.08	84.62	76.92	50.00	80.77	73.077
Running time (s)	0.0055	0.0124	0.0097	0.0073	0.0064	0.0083
Applying Feature Ranking with $k = 70$ features						
Dimension of input space	70	70	70	70	70	70
Best accuracy on the test set (%)	65.38	84.62	80.77	46.15	76.92	65.38
Running time (s)	0.0075	76.9230	0.0046	0.0027	0.0060	0.0062

Table 32: *Table of results in SVM classsifier after applying Feature Ranking*

Observation:

- Both methods have done a good job of reducing the data dimension and *increasing the performance of the model*, which can be demonstrated by running time.
- However, it is easy to see that the number of input data dimensions is *too high* or *too low* can cause unexpected results when implemented with the SVM classifier. This can be explained by the difficulty for SVM to find the optimal separation lines between classes when the dimension is too low or too high.
- Because the nature of PCA is related to the eigenvalues of a matrix, so the number of features after applying dimensionality reduction in the input matrix is always limited by the number of our samples (since the number of samples << the number of attributes of samples) so that the number of features that can be retained after applying PCA is always low (less than or equal to 130, is the number of features of input data points).
- The best result with PCA is that with **54 input features retained** along with using **blocked LBP extraction for grayscale images**, we achieve an accuracy of **96.15%**. That is, even though the data dimension was reduced a lot, the important information is still retained, enabling to keep the accuracy of the prediction almost absolute.

When using PCA or Feature ranking in this problem, we want to retain the important information in the human faces in the images. However, the result of feature ranking in the above cases is not good. Perhaps we need to test more parameter values. Finally, after long tests with looking at the accuracy change when changing parameters to find the extreme points, we have found a way to achieve **100%** accuracy which is the absolute value we want. We reach this absolute value using feature ranking, the reason why using this method is that the number of retained features that can be kept by using feature ranking is much wider, not limited like PCA. Here is the result:

- When using the data extraction that is *a combination of LBP and HOG*, also using the *Feature Ranking data dimensionality reduction at the lowest parameter* $k = 0.067$ (i.e. the ratio of the number of features we retain compared to the entire feature), we obtained a new dataset with dimensionality of 1121. The result of *the SVM algorithm with the kernel is linear* (the best kernel for our dataset, this is derived from throughout the experiment) using that input vectors gave **100%** accuracy results.
- On the other hand, when using *LBP feature extraction for block color images with size* 32×32, then continue to use *Feature Ranking data dimensional reduction at parameter level* $k = 1025$ (i.e. the number of features we retained), we have achieved a new dataset with 1025 dimensions, then using *the SVM version as above* gave an absolute result **100%**.

$\Longrightarrow$ The reduction of the data dimension using the Feature Ranking algorithm go in the way that have been proposed. That is, we have retained the most characteristic features of each person's face, on the one hand, both make the data points better clustered and remove the unimportant attributes that interfere with the classifier in the process. This is the most important reason why we can get the alsolute result.

And for sure, in machine learning, a certain algorithm only makes sense for a particular dataset. That means no algorithm is the best and fixed, it is certain that on more difficult datasets we will have to experiment more and combine methods more effectively to find our solutions.

4.2 Applying PCA for article classification

With the article classification using PCA, we mentioned above, so we probably won't discuss it here anymore.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Tiep Vu Huu. "Principal Component Analysis - Part 1". In: (Jun 15, 2017). URL: `https://machinelearningcoban.com/2017/06/15/pca/`.

[3] Zeren D. Yenice et al. *SPSA-FSR: Simultaneous Perturbation Stochastic Approximation for Feature Selection and Ranking*. 2018. arXiv: `1804.05589 [stat.ML]`.