Image Classification Using SVM, KNN and Performance
Comparison with Logistic Regression

Team member: Qi Gu, Zhifei Song

Abstract: Image classification is one important branch of Artificial intelligence; its application seems a promising direction in the development of search engine (We search information by input an image other than some text). There are lots of different machine learning algorithms used for image classification nowadays. The goal of this project is investigating two of the most popular machine learning algorithms: KNN and SMO [1], then implementing them in Java, we will also call Logistic regression algorithm for algorithm comparison. After that, we will discuss the performance of each algorithm above for image classification based on drawing their learning curve, selecting different parameters (KNN) and comparing their correct rate on different categories.

I. Success Metrics
We will consider our project successful if we accomplish the following:
1. Implement completely SVM and KNN which means our classifications can be easily run by the users and succeed in learning training data and classifying testing data with no worse performance than the Logistic Regression classification in Weka’s classification package [7]. We include our java program files (2 folders for SVM and KNN), dataset and readme.txt (how to use) in project’s zip package.
-COMPLETE

2. We design three groups of experiment to test the performance of our classifications and analysis the reason for the results.
-COMPLETE

II. Introduction to image classification:
The term image classification refers to the labeling of images into one of a number of predefined categories. Although it is seemed not a very difficult task for humans, it has proved to be a difficult problem for machines. Therefore, image classification challenge is also used for image-based CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart). To address image-based CAPTCHA problem and other problems like web searching, surveillance and sensor system, Designing and implementing automatic image classification algorithms has been an important research field for decades.

III. Data
The image data we use to do the classification is downloaded from [5]. We involve 3 categories in our experiment: African people, Flowers and Fashion, each of them containing 100 positive images, and 100 negative images (randomly collected from other categories). All images are in JPEG format with size $384 \times 256$ or $256 \times 384$. Here are some samples from each image category:

Then we transfer each image into a feature vector and write them into a single data file in specific format. The image feature vectors are generated by Qi’s another project, which first cluster the image into 16 regions, each region is represented by 6 attributes: 3 for color (LUV) and 3 for texture (wavelet transformation). Then use optimization method to learn the region prototypes that represent similar regions frequently occur in the positive training set. Finally, map the image into this region prototype space. The number of region prototypes varies from different categories basically depends on the complexity of the image layout of a category. Therefore, different category may have different feature vector length. In our experiment, the vector lengths of three categories are 80, 19 and 69.

IV. Detailed information of algorithms.

1. KNN
   (1) Definition
   KNN stands for “k-nearest neighbor algorithm”, it is one of the simplest but widely using machine learning algorithm. An object is classified by the “distance” from its neighbors, with the object being assigned to the class most common among its k distance-nearest neighbors. If $k = 1$, the algorithm simply becomes nearest neighbor algorithm and the object is classified to
the class of its nearest neighbor.

Distance is a key word in this algorithm, each object in the space is represented by position vectors in a multidimensional feature space. It is usual to use the Euclidean distance to calculate distance between two vector positions in the multidimensional space.

(2) Algorithm analysis
Draw votes avoidance: In a two-class classification problem (our experiment is exactly such kind of problem), we would want to choose \( k \) to be an odd number to avoid a draw votes.

The training process for KNN consists only of storing the feature vectors and class labels of the training samples. [2] One major problem to using this technique is the class with the more frequent training samples would dominate the prediction of the new vector, since they more likely to come up as the neighbor of the new vector due to their large number. To address this problem, in our experiment, we use the same number (100) image for each class.

K-selection, another important problem we should take into account is how to choose a suitable \( K \) for this algorithm. Generally, according to Shakhnarovish et.al[3], larger values of \( k \) reduce the effect of noise on the classification, but make boundaries between classed lsee distinct. Choosing an appropriate \( K \) is essential to make the classification more successful.

2. SMO (Sequential Minimal Optimization)
SMO is a new SVM learning algorithm which is conceptually simple, easy to implement, often faster and has better scaling properties than a standard SVM algorithm. To explain what SMO algorithm is, we first exam the standard SVM algorithm:

(1) SVM (Support Vector Machine) definition
Viewing input data as two sets of vectors in an n-dimensional space, a SVM will construct a separating hyperplane in that space, which maximizes the margin between the two data sets [4].

In the case of support vector machine, an object is viewed as a n-dimensional vector and we want to separate such objects with a n-1 dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data, we also interested in finding out whether we can get the maximum margin between the two data sets. (figure 1) The goal of SVM is try to address the nearest distance between a point in one class and a point in the other class being maximized and draw a hyperplane to classify two categories as clearly as possible.
The above figure shows 3 Hyperplanes in 2-Dimentional space. H3 does not separate the two classes, H1 does, with a small margin and H2 with the maximum margin. The goal of SVM is trying to find H2.

For implementing SVM on image classification, we are given a certain number p of training data, each data has two parts: the n-dimensional vector of image features and the corresponding labels of data (either 1 or -1)

$$S = \{(x_i,y_i)\mid x_i \in \mathbb{R}^n, y_i \in \{-1,1\}\}_{i=1}^p$$

Each $x_i$ is a n-dimensional vector. SVM want to give out the maximum-margin hyperplane dividing the objects with label $(y_i) = 1$ from those with label $= -1$.

(2) SMO algorithm
SVM are starting to enjoy increasing adoption in the machine learning communities, but two major weaknesses of it limited use by engineers. First the training of SVM is slow, especially for large problems. Second, SVM training algorithms are complex, subtle and sometimes difficult to implement.

The QP problem to train an SVM is shown below: we need to find suitable Lagrange multipliers $\alpha_i$ to get the following function reach its maximum value.

$$\max W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \langle x_i, x_j \rangle \alpha_i \alpha_j \quad \text{Subject to} \quad \forall i, 0 \leq \alpha_i \leq C$$
\[ \sum_{i=1}^{n} y_i \alpha_i = 0 \]

Function 1

(3) SMO strongpoint

Unlike standard SVM learning algorithms, which use numerical quadratic programming as an inner loop, SMO uses an analytic QP step. Because SMO spends most of its time evaluating the decision function, rather than performing QP.

The QP problem in equation 1 is solved by the SMO algorithm. A point is an optimal point of function 1 if and only if the KKT (Karush-Kuhn-Tucker) conditions are fulfilled. The KKT conditions can be shown as follows: for all \( i \):

\[
\begin{align*}
\alpha_i = 0 & \Rightarrow y_i f(x_i) \geq 1, \\
0 < \alpha_i < C & \Rightarrow y_i f(x_i) = 1, \\
\alpha_i = C & \Rightarrow y_i f(x_i) \leq 1.
\end{align*}
\]

Function 2

The KKT conditions can be evaluated one example for one Lagrange multiplier once, which is useful in the construction of the SMO algorithm.

At every step, SMO chooses two Lagrange multipliers to jointly optimize (using KKT condition above), finds the optimal values for these multipliers and updates the SVM to reflect the new optimal values. The advantage of SMO lies in the fact that solving for two Lagrange multipliers can be done analytically. An entire inner iteration due to numerical QP optimization is avoided. In addition SMO does not require extra matrix storage (to store previous \( \alpha_1, \alpha_2 \) and current \( \alpha_1, \alpha_2 \) we only need 2*2 matrices). Therefore, very large SVM training problems can fit inside of the memory of a personal computer.

The two Lagrange multipliers must fulfill all of the constraints of the full problem. The inequality constraints (function 1) cause the Lagrange multipliers to lie in the box (Figure 2). Therefore, one step of SMO must find an optimum of the objective function on a diagonal line segment. In figure 2, \( \gamma = \alpha_1 \text{(old)} + s \alpha_2 \text{(old)} \), is a constant that depends on the previous value of \( \alpha_1 \) and \( \alpha_2 \), \( s = y_1 y_2 \).
V. Code design:

**Input Data**: Implemented in method `readData(file)`
Parse given data file containing image feature vectors which are written in specific format (one line for one vector, attributes are separated by comma, class identity is the last attribute). Store the feature values in `featureVector[]` and class value in `target[]`. Return sample number.

**Train SVM**: Implemented in program file `training.java`
Read in the training data by calling `readData(train file)`. Loop over training examples to select multipliers to optimize. Stop until no more multipliers need to be optimized. Each step, SMO will use first heuristic to choose the multiplier that has violated the KKT conditions. Here, KKT condition is satisfied when function2 is fulfilled.
Then pass this example to function `examineExample(example i)` using second heuristic to find another multiplier and optimize these Lagrange multipliers jointly.

**Choose Multipliers to Optimize**: Implemented in method `examineExample(example i)`
Chooses the second Lagrange multiplier to maximize the size of the step then alter two Lagrange multipliers to move uphill in the objective function `learnFunc(example k)`. SMO approximates the step size by $|E_1 - E_2|$ when it keeps a cached error value $E$ in `errorCache[]` for every $(0 < \alpha_i < C)$ in the training set and then chooses an error to approximately maximize the step size. If $E_1 > 0$, SMO chooses an example with minimum error $E_2$. If $E_i < \gamma$, then $y_1 = y_2 \Rightarrow \alpha_1 - \alpha_2 = \gamma$.
0, it chooses an example with maximum error $E_2$. When a Lagrange multiplier is involved in a joint optimization, its cached error is set to zero. When a joint optimization occurs, the stored errors for all multipliers $\alpha_k$ that are not involved in the optimization are updated according to $E_k^{new} = \text{learnFunc}(k) - y_k$. If the above heuristic doesn’t work, then SMO will first iterate through the non-bound ($0 < \alpha_i < C$) examples, searching for a second example that can make positive progress; if none of the non-bound examples make positive progress, then it starts iterating through the entire training set until find one. When an example $i$ is found, it will call takeStep($I, k$) to make a positive progress.

**Optimize Two Multipliers:** Implemented in method takeStep(multiplier i, multiplier j)

Solve for the two Lagrange multipliers. SMO first computes the constraints on these two multipliers and then solves for the constrained maximum. Under normal circumstances, there will be a maximum along the direction of the linear equality constraint, when $\eta = 2 kernalFunc(x_i, x_j) - kernalFunc(x_i, x_i) - kernalFunc(x_j, x_j) < 0$. In this case, SMO computes the maximum along the direction of the constraint:

$$\alpha_j^{new} = \alpha_j^{old} - \frac{y_j(E_i - E_j)}{\eta}.$$

$$\alpha_i^{new} = \alpha_i^{old} + y_i y_j (\alpha_j^{old} - \alpha_j^{new})$$

Function 3

Under unusual circumstances $\eta$ will not be negative, A zero $\eta$ can occur if more than one training example has the same input vector, if so, only those terms in the objective function that depend on $\alpha_j$ need be evaluated.

**Objective Function:** implemented in method learnFunc(example i)

Calculate the objective function by the following equation:

$$f = \sum \alpha_k y_k (\text{kernelFunc}(k,i)) \text{ (for all example k in the training set)}$$

**Kernel Function:** implemented in kernelFunc(example i, example j)

Use rbfKernel to calculate kernel function, return rbfKernel value for example i and j.

**Using a Radial Basis Function as Kernel:** implemented in method rbfKernel(example i, example j)

Normally a Gaussian will be used as the RBF, the output of the kernel is dependent on the Euclidean distance of $x_j$ from $x_i$ (one of these will be the support vector and the other will be the testing data point). The support vector will be the centre of the RBF and will determine
the area of influence this support vector has over the data space. So, RBF kernel is:

\[ K(x_i, x_j) = \exp\left( -\frac{\|x_i - x_j\|^2}{2\sigma^2} \right) \] [6]

**Output SVM**: implemented in method `writeSvm` *(printstream)*
Print out the learned SVM model in the specific format, including multipliers and training feature vectors.

**Input SVM**: implemented in method `readSvm(file)`
Read in a learned SVM model, store multipliers in `alph[]` and training feature vectors in `featureVector[]`

**Classify the Test File**: implemented in program file `classifying.java`
Read in the testing data by calling `readData(test file)`. Loop over testing examples and use objective function to predict the target. If the predicted target is not same as the real target, then increase the error. Print out the precision at the end of the program.

**VI. Experiment and Analysis:**

1. **KNN algorithm discussion with different k numbers.**
In the following 3 experiments, we choose 200 images as input dataset: 100 positive images (fashion, flower or African people), 100 negative images, then randomly select 160 images from the dataset (80 positive, 80 negative) as training set, and 40 images (20 positive, 20 negative) as testing set. Then run KNN with different k values. We choose odd number for k values for the reason we discussed in section III.

![KNN accuracy for different k value](image)

**Figure 3**: KNN accuracy for different k value under Fashion category
Analysis: The above 3 figures share one characteristic in common: with the k value ascending, the accuracy rate for KNN first increases then decreases, this is because larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct.

Comparing figure 4 with figure 3 and 5, we find the KNN classification accuracy rate is higher for flowers than the other two. In figure 4, the highest accuracy is reached very soon when k is still a smaller number (k=5) and then decreased gradually while for fashion and African categories, the highest accuracy are reached slowly (k=21 for African, k = 48). That’s probably because Flowers feature vectors are more “dense” in the multidimensional space than the other two.

We can also find an interesting phenomenon: the average accuracy for different dataset: Flowers > African People > Fashion, the value of k needed to reach the highest accuracy value: Flowers < African People < Fashion.
2. Learning curve analysis.
In the following 3 experiments, we choose 200 images as input dataset: 100 positive images (fashion, flower or African people), 100 negative images, then gradually increase the training set size.

![SMO learning curve](image)

Figure 6: SMO learning curve

![LR learning curve](image)

Figure 7: Logistic Regression learning curve
Analysis: In all above 3 figures, as the training set grows, the prediction quality increases, they are all happy graphs. One thing we want to mention here is the SMO learning curve does not seem that happy and the accuracy rate has not changed greatly from 2% training set to 50% training set. One possible reason is the SMO’s hyperplane selection is a slow adjusting course.

In figure 7, we can also see the classification accuracy rate decreases after choosing more than 92% data as training data, the reason is overfitting, that we use the resulting freedom to find meaningless “regularity” in the data.

3. Different classifiers performance comparison.

Analysis: In figure 9, we run different classifiers on different categories and compare their
performances. For all classifiers, we can find flowers classification accuracy rate is always the highest while fashion classification accuracy rate is always the lowest. This implies the vector features for flowers are more distinguishable than the other two.

In the aspect of classifier performance on the same dataset, for our database, the logistic regression algorithm performance is not as good as the other two which guarantee us of meeting our first success metric.

Reference:

[5]: http://john.cs.olemiss.edu/~ychen/DDSVVM