AMATH 582 Homework 4: Classifying Digits

Emma Nuss, github: emmashie

Abstract

Singular value decomposition and data classifying techniques were applied to a dataset of hand written digits. SVD analysis shows data dimensionality could be reduced and data types had distinct clusters. Data classifying techniques proved effective at correctly identifying two digits apart. The linear discriminant analysis (LDA) method was least effect with notable digits having low success rates, while the decision tree classifier (DTC) and the support vector machine (SVM) methods had high success rates for all digits. These methods also proved effective at classifying all 10 digits, with DTC and SVM having higher success rates over LDA.

1. Introduction

The aim of this project is building a data classifying algorithm that correctly identifies the handwritten digit in images. To understand the dimensionality and variance of the data, singular value decomposition (SVD) was applied to two datasets of images of written digits. After exploring the data, three types of data classifying algorithms (linear discriminant analysis, decision tree classifiers, and support vector machines) were applied to the digit data. The accuracy of these classifiers were tested and compared with each other. The theory behind these classifiers, their implementation, and the results of this analysis is described below.

2. Theoretical Background

2.1. Classifying Data

A large area in data science today is data classifying categories of data by using mathematical concepts and statistical information to differentiate types of data in a large dataset. Each method for data classifying is unique, but the basic algorithm has the same construction. First, a training dataset with known data types is fed into a statistical model that 'learns' the statistical patterns associated with each of the N data types. A test dataset, with the same data types, can be put into this statistical model and the type of each point in the dataset with be classified into one of the N data types.

Linear disccriminant analysis (LDA) is a data classifying technique which reduces high-dimensional data and projects the data to lower dimension and then finds a line to discriminate between the data types. LDA relies on understanding the scatter of data within each data type and between data types. In an ideal dataset, each data type would have low variance within the type, but high variance between types i.e. each data type would be closely clustered together but far away from other data type clusters. In an ideal case with two data types, the data would be projected onto a line and easily be separated with a decision line between the two types of data.

Finding the best projection for the data to make this discriminant line is an optimization problem. The optimal projection is found using:

$$w = \text{argmax} \frac{w^T S_B w}{w^T S_W w} \tag{1}$$

where S_B is the between-class scatter and S_W is the within-class scatter and are given by:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{2}$$

$$S_W = \sum_{j=1}^{2} \sum_{x} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T \tag{3}$$

and solving the eigenvalue problem $S_B w = \lambda S_W w$ (Kutz 2013).

There are many other types of data classifiers, two classifiers that will be used in our analysis are a decision tree classifier (DTC) and a support vector machine (SVM). A support vector machine is similar to LDA in the sense that an optimal line between clusters of data is found. For SVM this line is the optimal separator, rather than the optimal projection like in LDA. For a dataset with two data types, this optimal separator is found in SVM through an optimization problem that finds the line where there is maximal separation between the data. In contrast, a decision tree classifier, as its name suggests, uses a decision tree as a predictive model. The model is trained on a known dataset and identifies points in the data that split the classifications of data types most efficiently. When the model is applied to a test dataset, a series of decisions are evaluated to categorize the data into data types.

3. Algorithm Implementation and Development

3.1. Data

The mnist dataset of written digits images was loaded in from the scikit-learn python package. Two image datasets are included in the mnist dataset, the training dataset of 60,000 images of digits 0 to 9 and the test dataset of 10,000 images of digits 0 to 9.

3.2. Singular Value Decomposition & Dimensionality Reduction

Both the training and test data were reshaped into a data matrix where each image (28×28 pixels) was reshaped into an array of length 784, the mean was subtracted out, and then input into the columns of the data matrix. SVD of each data matrix was completed and the percentage of variance explained by each mode was computed by squaring and normalizing the diagonal elements of the Σ matrix. In addition, the mode at which the percentage of the variance explained fell below 5% of the variance explained by mode 1 was identified. Images were reconstructed using the first n modes that were above this 5% of the max variance threshold. In addition, three columns of $\mathbf{V}$ from the SVD were plotted to visualize the projection of the data onto the $\mathbf{V}$ modes and visualize the clustering of the data.

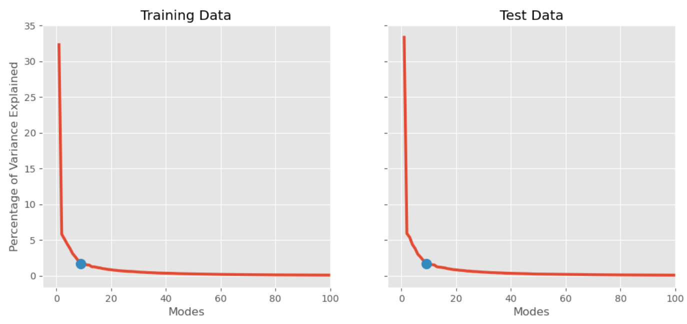

Figure 1: SVD spectrum for the training and test mnist data. The blue dot denotes the mode (10) at which the variance falls below 5% of the maximum variance.

3.3. *Classifying Data*

After a failed attempt to convert and modify the 'dc_trainer' Matlab code in Kutz 2013 to Python, a data classifying algorithm using machine learning functions in the scikit-learn Python package was built. First, the data matrices of training and test data that were compiled with means removed for the SVD analysis were sub-sampled to data that only corresponded to two digits. Then the sub-sampled data matrix was fed into three different data classifying functions.

The first data classifying function was LDA. For LDA, the data was first standardized using the 'StandardScaler' function to scale the data so that the standard deviation is equal to 1. Next, the training data was fed into the LDA function in scikit-learn. Then, the test data was classified using the LDA model found from the training data. A success rate was computed by the number of correct classifications divided by the total number of test data. This same process was then used again, without the scaling, but the decision tree classifier function was used in place of LDA. And lastly, that process was repeated with the support vector machine function.

These three data classifying algorithms were computed for all combinations of digits and the success rates were compiled into a matrix for visualization. Low success and high success combinations of digits were identified from the success matrix. Two sub-sampled data matrices corresponding to three low success digits and three high success digits were compiled and run

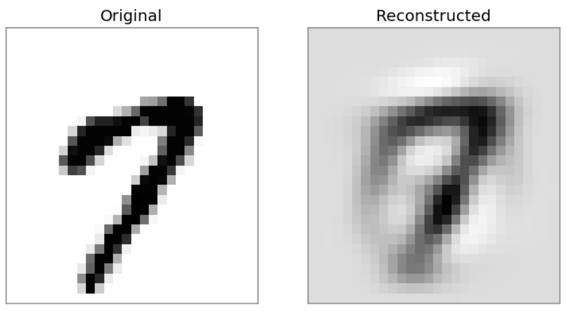

Figure 2: A sample image of the original and reconstructed image using 10 modes.

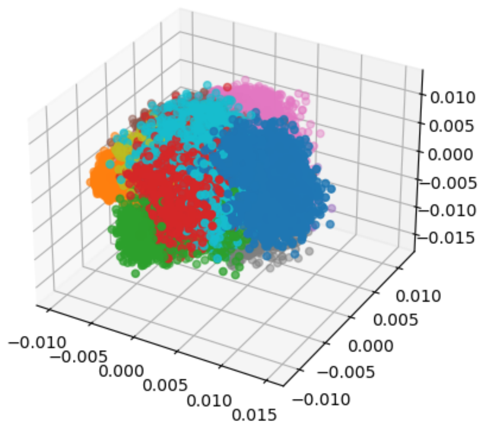

Figure 3: Scatter plot of **V** modes 2, 3, and 5. Data are colored by digit type with: 0 = blue, 1 = orange, 2 = green, 3 = red, 4 = purple 5 = cyan, 6 = grey, 7 = pink, 8 = olive, 9 = brown

through the same classifying algorithm. The success rates were compiled and compared between classifier types. Lastly, this data classifying algorithm was run another time but with the entire training dataset. Success rates for the three classifier types were computed and compared.

Additionally, the training data was run through the three types of classifying algorithms to test how well the models do on the training dataset versus the test dataset. This comparison was done for the 3 digit and 10 digit cases, but not the 2 digit cases since all the combinations of 2 digit cases was computationally slow and the 3 and 10 digit cases provided sufficient comparison.

4. Computational Results

4.1. Singular Value Decomposition & Dimensionality Reduction

For both the training and test datasets the SVD spectrum shows that about 30% of the variance is explained in the first mode with the percentage of variance falling to 5% of the maximum variance by mode 10 (Fig. 1). For both datasets, approximately 60% of the variance is explained by the first 10 modes. Images are reasonably well reconstructed using only the first 10 modes (Fig. 3). Lastly, for modes 2, 3, and 5, the digits can be seen as clusters in the data scatter suggesting that each digit has relatively distinct properties and can be identified using data classifier algorithms.

4.2. Classifying Data

Overall, data classifying methods using LDA, DTC, and SVM proved generally effective at identifying all combinations of two digits apart in the test data. SVM had extremely high success rates (> 0.95) for all combinations of two digits, while DTC had high success rates (> 0.9) for all combinations of digits (Fig. 4). With LDA, there were notable combinations of digits with low success rates (~0.5 - 0.7) while most other combinations of digits has relatively high success rates (Fig. 4). The notable low success rate combinations with LDA were 8 vs 5, 8 vs 3, 8 vs 2, 5 vs. 3 and 9 vs. 4.

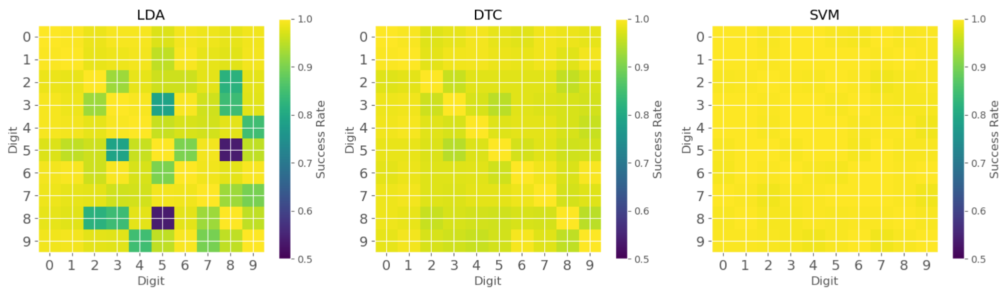

Figure 4: Success rates for all two digit combinations for LDA, DTC, and SVM. Note that data classifying was not computed for the diagonal elements and are set to 1 and the off diagonal components are symmetric since digit i vs. digit j is the same as digit j vs. digit i.

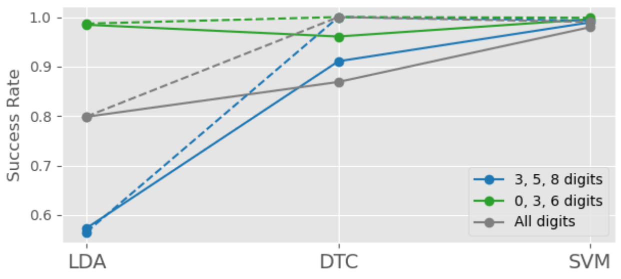

Figure 5: Success rates for two 3 digit combinations and all digits for LDA, DCT, and SVM. Solid lines denote success rates from using test data and dashed lines denote success rates when using the training data.

For classifying more than one digit in the test data, the LDA method continued to be the weakest classifier, with DTC and SVM generally having higher success rates across the board. DTC and SVM both had a smaller range in success rates between the low success rate combination of digits, the high success rate combination of digits, and all digits, whereas the LDA method had low success (0.58) with the ”hard” combination, higher success (0.8) with all digits, and the highest success rate (0.98) with the ”easy” digits.

Running the training data through the LDA and SVM models gives very similar results as with the test data. For the two 3 digit cases and all 10 digits, the success rates were roughly identical as with the test data for these two methods (Fig. 5). For the DTC, the training data gives a success rate of 1 for all three cases (Fig. 5). The decision tree that is generated by the DTC method is not constrained in the number of decision links, so the tree is able to classify every single training data point correctly. If we were to limit the decision links, this success rate would go down. In a ”real world” scenario, likely we would want to do this so that our model is more generalizable to many datasets.

5. Summary and Conclusions

SVD and data classifying techniques are useful tools to reduce dimensionality in your data and identify data characteristics. Data classifying techniques proved effective at correctly identifying digits apart. The LDA method was least effect, while the DTC and the SVM methods had high success rates across the board. This pattern was true for classifying two digits, three digits, and all ten digits.

Appendix A Python Functions

Relevant Python functions:

- `[U, Sdiag, VH] = np.linalg.svd(X)`: Computes the SVD of matrix X where Sdiag is the diagonal components of Σ and VH is the transpose of $\mathbf{V}$
- `StandardScaler()`: Scales data by standard deviation
- `clf = LDA(solver='svd')`: Creates a LDA model using SVD
- `dt = DecisionTreeClassifier()`: Creates a DTC model
- `clf = svm.SVC()`: Creates a SVM model
- `x = np.arange(0, n)`: Creates a 1D array of integers starting at 0 up to `n-1`
- `z = np.append(x, y)`: Appends array y to the end of array x and stores it in variable z
- `ind = np.argwhere(D == condition) or np.where(D == condition)`: Finds the indices of an array where a logical condition is satisfied
- `np.asarray(D)`: Takes a list or some non-array and converts it to an array type
- `L = [i for i in range(n)]`: (List comprehension) Creates a list of `0` to `n-1`, can be used for high dimensions of lists and with if conditional statements
- `os.path.join(path, filename)`: Creates a path object based on the `path` and `filename` given

Appendix B Python Code

Code can be found at: https://github.com/emmashie/amath-582

```python
#!/usr/bin/env python
""" functions used for AMATH 582 hw4
"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn import svm

def svd(data, data_axis=1, alpha=0.05):
        [nimgs, nx, ny] = np.shape(data)
        #X = data - np.expand_dims(np.mean(data, axis=data_axis), axis=data_axis)
        X = np.reshape(data, (nimgs, nx*ny), order='F')
        X = X - np.expand_dims(np.mean(X, axis=0), axis=0)

        U, Sdiag, VH = np.linalg.svd(X.T, full_matrices=False)
        V = VH.T
        Snorm = 100*Sdiag**2/np.sum(Sdiag**2)
        maxind = np.argmin(abs(Snorm-alpha*Snorm[0]))

        Xrec = np.matmul(np.matmul(U[:,:maxind], np.diag(Sdiag[:maxind])), VH[:maxind,:])
        Xrec = np.reshape(Xrec, (nx, ny, nimgs), order='F')
        return U, Sdiag, V, X, Snorm, maxind, Xrec

def lda(X_train, label_train, X_test, label_test):
        for i in range(len(X_train)):
                X_train[i,:] = np.squeeze(StandardScaler().fit_transform(X_train[i,:].reshape(-1,1)))
        clf = LDA(solver='svd')
        clf.fit(X_train, label_train)

        X_transform = clf.transform(X_train)

        y_predict = np.copy(label_test)
        for i in range(len(label_test)):
                y_predict[i] = clf.predict(X_test[i,:].reshape(1,-1))[0]
        matches = np.where(y_predict==label_test)[0]
        return len(matches)/len(y_predict)

def dct(X_train, label_train, X_test, label_test):
        dt = DecisionTreeClassifier()
        dt.fit(X_train, label_train)
        y_predict = dt.predict(X_test)
        matches = np.where(y_predict==label_test)[0]
        cm = confusion_matrix(label_test, y_predict, labels=np.unique(label_test))
        cm_norm = cm/np.expand_dims(np.sum(cm, axis=0), axis=0)
        cm_norm2 = cm/np.expand_dims(np.sum(cm, axis=1), axis=1)
        return len(matches)/len(y_predict)

def svc(X_train, label_train, X_test, label_test):
        clf = svm.SVC()
        clf.fit(X_train, label_train)
        y_predict = np.copy(label_test)
        for i in range(len(label_test)):
                y_predict[i] = clf.predict(X_test[i,:].reshape(1,-1))[0]
        matches = np.where(y_predict==label_test)[0]
        return len(matches)/len(y_predict)

def compare_classify(X_train, label_train, X_test, label_test, digits):
        ind = []
        for digit in digits:
                dum = np.where(label_train==digit)[0]
                for i in range(len(dum)):
                        ind.append(dum[i])
        ind_test = []
        for digit in digits:
                dum = np.where(label_test==digit)[0]
                for i in range(len(dum)):
                        ind_test.append(dum[i])
        lda_success = lda(X_train[ind,:], label_train[ind], X_test[ind_test,:], label_test[ind_test])
        dct_success = dct(X_train[ind,:], label_train[ind], X_test[ind_test,:], label_test[ind_test])
        svm_success = svc(X_train[ind,:], label_train[ind], X_test[ind_test,:], label_test[ind_test])
        return lda_success, dct_success, svm_success
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from scipy import linalg
import os
import hw4_functions as f
plt.ion()
plt.style.use('ggplot')

(img_train, label_train), (img_test, label_test) = mnist.load_data()
[nimgs_train, nx, ny] = np.shape(img_train)
[nimgs_test, nx, ny] = np.shape(img_test)

### svd of training and test data
U_train, Sdiag_train, V_train, X_train, Snorm_train, maxind_train, Xrec_train = f.svd(img_train, data_axis=1, alpha=
U_test, Sdiag_test, V_test, X_test, Snorm_test, maxind_test, Xrec_test = f.svd(img_test, data_axis=1, alpha=0.1)

#Xrec_train = np.reshape(Xrec_train, (len(X_train[0,:]), len(X_train))).T

#X_train = Xrec_train
### two digit compare
labels = np.unique(label_train)
success_rates_lda = np.ones((len(labels),len(labels)))
success_rates_dct = np.ones((len(labels),len(labels)))
success_rates_svm = np.ones((len(labels),len(labels)))
for i in range(len(np.unique(label_train))):
        for j in range(len(np.unique(label_train))):
                if i!=j:
                        digits = [i, j]
                        success_rates_lda[i,j], success_rates_dct[i,j], success_rates_svm[i,j] = f.compare_classify(
        print('done with %d' % i)


### three digit compare
digits_hard = [3, 5, 8]
success_rates_lda_3hard, success_rates_dct_3hard, success_rates_svm_3hard = f.compare_classify(X_train, label_train,
digits_easy = [0, 3, 6]
success_rates_lda_3easy, success_rates_dct_3easy, success_rates_svm_3easy = f.compare_classify(X_train, label_train,
### all digit compare
success_rates_lda_all, success_rates_dct_all, success_rates_svm_all = f.compare_classify(X_train, label_train, X_tes
print('done with 3 and 10 digit with test data')

digits_hard = [3, 5, 8]
train_success_rates_lda_3hard, train_success_rates_dct_3hard, train_success_rates_svm_3hard = f.compare_classify(X_t
digits_easy = [0, 3, 6]
train_success_rates_lda_3easy, train_success_rates_dct_3easy, train_success_rates_svm_3easy = f.compare_classify(X_t
### all digit compare
train_success_rates_lda_all, train_success_rates_dct_all, train_success_rates_svm_all = f.compare_classify(X_train,



if 1: ## plot svd spectrum
        plt.style.use('ggplot')
        fig, ax = plt.subplots(ncols=2, sharey=True, figsize=(12,5))
        ax[0].plot(np.arange(1,len(Snorm_train)+1,1), Snorm_train, linewidth=3)
        ax[0].plot(maxind_train, Snorm_train[maxind_train], 'o', markersize=10)
        ax[0].set_ylabel('Percentage of Variance Explained')
        ax[0].set_xlabel('Modes')
        ax[0].set_title('Training Data')
        ax[0].set_xlim(-5,100)

        ax[1].plot(np.arange(1,len(Snorm_test)+1,1), Snorm_test, linewidth=3)
        ax[1].plot(maxind_test, Snorm_test[maxind_test], 'o', markersize=10)
        ax[1].set_xlabel('Modes')
        ax[1].set_title('Test Data')
        ax[1].set_xlim(-5,100)
        fig.savefig('svg_spectrum.png')

if 1: ## plot reconstructed images
        img_num = 15
        fig, ax = plt.subplots(ncols=2, figsize=(9,4.5))
        ax[0].pcolormesh(img_train[img_num,::-1,:], cmap='Greys')
        p = ax[1].pcolormesh(Xrec_train[::-1,:,img_num], cmap='Greys')
        #p.set_clim(0,np.max(Xrec_train[img_num,:,:]))
        ax[0].get_xaxis().set_visible(False)
        ax[0].get_yaxis().set_visible(False)
        ax[1].get_xaxis().set_visible(False)
        ax[1].get_yaxis().set_visible(False)
        ax[0].spines['bottom'].set_color('0.5')
        ax[0].spines['top'].set_color('0.5')
        ax[0].spines['right'].set_color('0.5')
        ax[0].spines['left'].set_color('0.5')
        ax[1].spines['bottom'].set_color('0.5')
        ax[1].spines['top'].set_color('0.5')
        ax[1].spines['right'].set_color('0.5')
        ax[1].spines['left'].set_color('0.5')
        ax[0].set_title('Original')
        ax[1].set_title('Reconstructed')
        fig.savefig('img_recon.png')

if 0: ## plot v projection scatter
        plt.style.use('default')
        label_train_str = np.unique([str(label) for label in label_train])
        colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:cyan', 'tab:grey', 'tab:pink'
        label_colors = [colors[label] for label in label_train]
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.scatter(V_train[:,1], V_train[:,2], V_train[:,4], c=label_colors)
        ax.legend(loc='best')
        fig.savefig('v_modes.png')

if 1:
        plt.style.use('ggplot')
        fig, ax = plt.subplots(ncols=3, figsize=(15,4))
        p0 = ax[0].imshow(success_rates_lda, cmap='viridis')
        fig.colorbar(p0, ax=ax[0], label='Success Rate')
        p0.set_clim(0.5,1)
        ax[0].set_xticks(range(len(labels)))
        ax[0].set_yticks(range(len(labels)))
        ax[0].set_xticklabels(labels, fontsize=14)
        ax[0].set_yticklabels(labels, fontsize=14)
        ax[0].set_title('LDA')
        ax[0].set_xlabel('Digit')
        ax[0].set_ylabel('Digit')

        p1 = ax[1].imshow(success_rates_dct, cmap='viridis')
        fig.colorbar(p1, ax=ax[1], label='Success Rate')
        p1.set_clim(0.5,1)
        ax[1].set_xticks(range(len(labels)))
        ax[1].set_yticks(range(len(labels)))
        ax[1].set_xticklabels(labels, fontsize=14)
        ax[1].set_yticklabels(labels, fontsize=14)
        ax[1].set_title('DTC')
        ax[1].set_xlabel('Digit')
        ax[1].set_ylabel('Digit')

        p2 = ax[2].imshow(success_rates_svm, cmap='viridis')
        fig.colorbar(p2, ax=ax[2], label='Success Rate')
        p2.set_clim(0.5,1)
        ax[2].set_xticks(range(len(labels)))
        ax[2].set_yticks(range(len(labels)))
        ax[2].set_xticklabels(labels, fontsize=14)
        ax[2].set_yticklabels(labels, fontsize=14)
        ax[2].set_title('SVM')
        ax[2].set_xlabel('Digit')
        ax[2].set_ylabel('Digit')
        fig.tight_layout()
        fig.savefig('2digit_compare.png')

if 1:
        plt.style.use('ggplot')
        fig, ax = plt.subplots(figsize=(7,3))
        ax.plot(np.arange(3), np.array([success_rates_lda_3hard, success_rates_dct_3hard, success_rates_svm_3hard]),
        ax.plot(np.arange(3), np.array([success_rates_lda_3easy, success_rates_dct_3easy, success_rates_svm_3easy]),
        ax.plot(np.array([success_rates_lda_all, success_rates_dct_all, success_rates_svm_all]), 'o-', color='tab:gr
        ax.plot(np.arange(3), np.array([train_success_rates_lda_3hard, train_success_rates_dct_3hard, train_success_
        ax.plot(np.arange(3), np.array([train_success_rates_lda_3easy, train_success_rates_dct_3easy, train_success_
        ax.plot(np.array([train_success_rates_lda_all, train_success_rates_dct_all, train_success_rates_svm_all]), '
        ax.legend(loc='best')
        ax.set_ylabel('Success Rate')
        ax.set_xticks(np.arange(3))
        ax.set_xticklabels(['LDA', 'DTC', 'SVM'], fontsize=14)
        fig.savefig('3digit_all_compare.png')
```