Recognizing Handwritten Digits

Jedidiah Harwood, Darya Petrov, Kevin Xu, Katya Katsy

Data Preprocessing

test_labels	10x4649
test_patterns	256x4649
train_labels	10x4649
train_patterns	256x4649

- USPS handwritten digit data
- 9298 total images (16x16)
- Training set: 4649 images
- Testing set: 4649 images
- The *_patterns variables 16x16 grey level pixel intensities, scaled to the range [-1, 1]
- The *_labels variables contain encode with values -1 and +1 of the classification with one +1 per column
- Possible labels 0-9

```
load('usps_resampled.mat');

%obtain all testing labels
testingLabels = [];
for i = 1:4649
    val = find(test_labels(:,i)==1);
    testingLabels(end+1) = val-1; %digits 0-9
end

%obtain all training labels
trainingLabels = [];
for i = 1:4649
    val = find(train_labels(:,i)==1);
    trainingLabels(end+1) = val-1;
end
```

```
%column indicies for each digit in train_labels
zeros_index = find(trainingLabels == 0);
ones_index = find(trainingLabels == 1);
twos_index = find(trainingLabels == 2);
threes_index = find(trainingLabels == 3);
fours_index = find(trainingLabels == 4);
fives_index = find(trainingLabels == 5);
sixs_index = find(trainingLabels == 6);
sevens_index = find(trainingLabels == 7);
eigths_index = find(trainingLabels == 8);
nines_index = find(trainingLabels == 9);
```

```
%columns for each digit
zero = train_patterns(:,zeros_index);
ones = train_patterns(:,ones_index);
twos = train_patterns(:,twos_index);
threes = train_patterns(:,threes_index);
fours = train_patterns(:,fours_index);
fives = train_patterns(:,fives_index);
sixs = train_patterns(:,sixs_index);
sevens = train_patterns(:,sevens_index);
eigths = train_patterns(:,eigths_index);
nines = train_patterns(:,nines_index);
```

```
%mean for each digit
means = zeros(256,10);
means(:,1) = mean(zero,2);
means(:,2) = mean(ones,2);
means(:,3) = mean(twos,2);
means(:,4) = mean(threes,2);
means(:,5) = mean(fours,2);
means(:,6) = mean(fives,2);
means(:,7) = mean(sixs,2);
means(:,8) = mean(sevens,2);
means(:,9) = mean(eigths,2);
means(:,10) = mean(nines,2);
```

Simple Classification Algorithm

- Convert 4969 training images (16x16 pixels) to $\vec{x}_1, \vec{x}_2, ..., \vec{x}_{4649} \in R^{256}$
- Convert 4969 testing images (16x16 pixels) to $\vec{y}_1, \vec{y}_2, ..., \vec{y}_{4649} \in R^{256}$

Using the Training Set:

- Calculate $\vec{u_i}$ for $i = (0, 1, ..., 9)$, the mean of all the $\vec{x}_j$ classified as digit i

Using the Testing Set:

- For each $\vec{y}_i$, classify the digit as k if it is closest to u_k
- Example of closeness measures: l2 norm (euclidean distance) or cosine distance

Results

Euclidean Distance Accuracy: 86.44%, Run Time: ~ (0.1 - 0.2) seconds
Cosine Distance Accuracy: 84.51%, Run Time: ~ (0.1 - 0.2) seconds

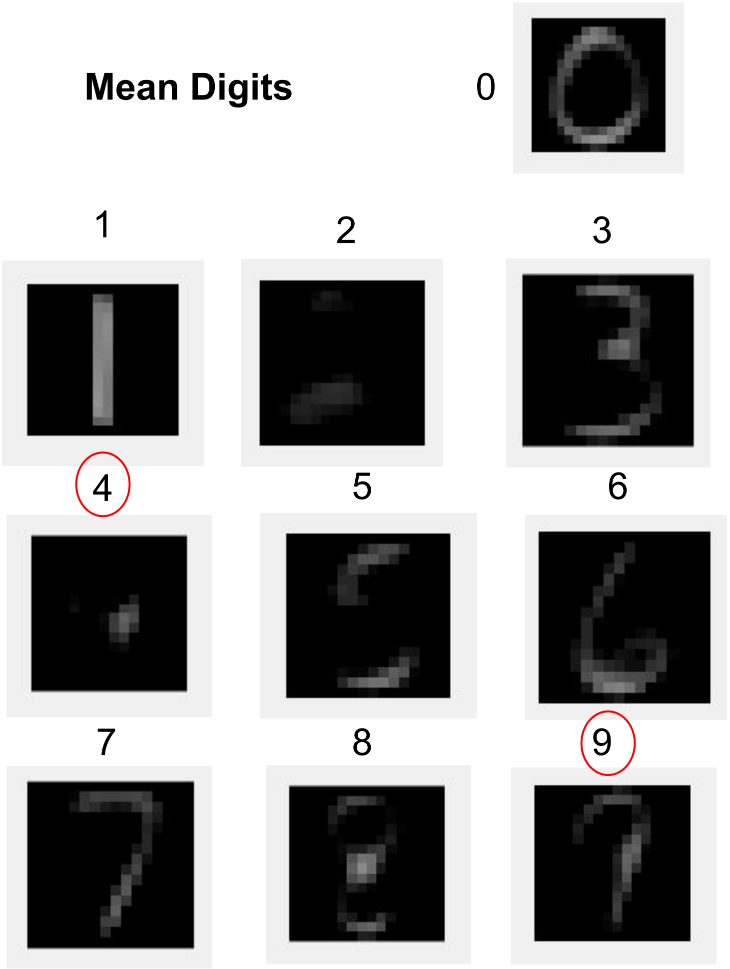
Flaws

- algorithm doesn't use any information about the variance of handwritten digits

```
%store classification as a 1x4649 array
test_class = NaN(1,4649);
tic %calculcate run time
%loop through every testing digit image
for i=1:4649
    %initialize large distance between matricies
    J = Inf;
    label = nan;
    %for each handwritten digit, find the closest mean
    for j=1:10
        %euclidean distance
        new_J = norm(test_patterns(:,i) - means(:,j));
        %cosine distance
%         a = test_patterns(:,i);
%         b = means(:,j);
%         new_J = 1 - (dot(a,b)/(norm(a)*norm(b)));
        if new_J < J
            J = new_J;
            label = j-1;
        end
    end
    %classify handwritten digit
    test_class(i) = label;
end
toc
```

Simple Classification Results

True Class \ Predicted Class	0	1	2	3	4	5	6	7	8	9
0	656	1	3	4	10	19	73	2	17	1
1		644		1			1		1	
2	14	4	362	13	25	5	4	9	18	
3	1	3	4	368	1	17		3	14	7
4	3	16	6		363	1	8	1	5	40
5	13	3	3	20	14	271	9		16	6
6	23	11	13		9	3	354		1	
7		5	1		7	1		351	3	34
8	9	19	5	12	6	6		1	253	20
9	1	15		1	39	2		24	3	314



- Da
- Go ..., $\vec{u}_{10}$
  to

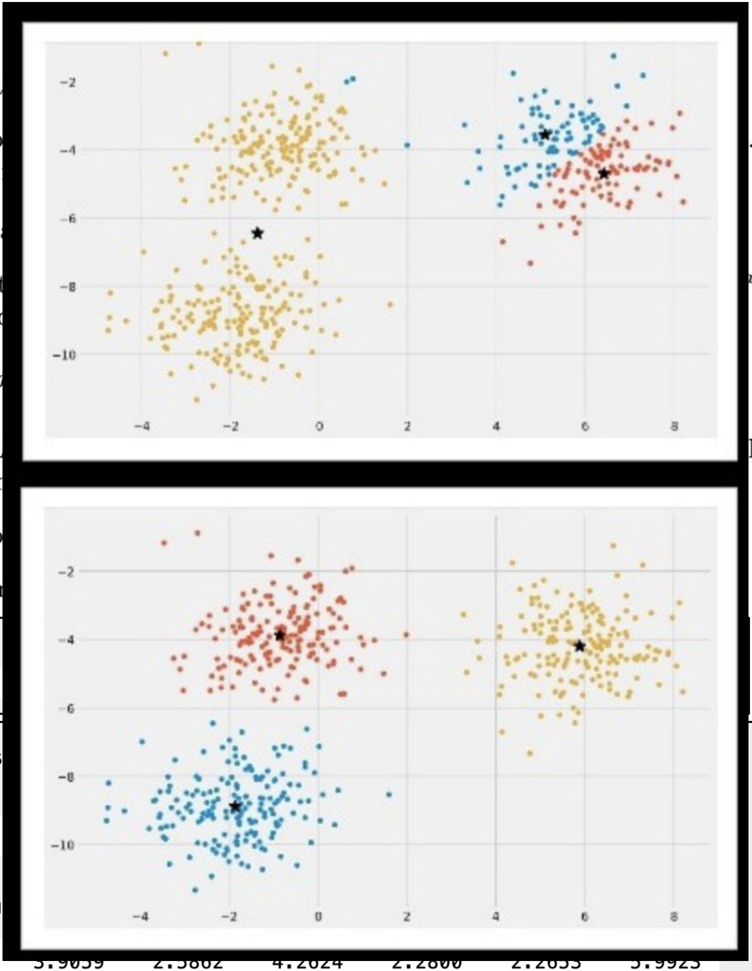
We c

- Let ... $x10$,
  and

  B_{ij}

- M ... hat
  co
- Go
- Kn

1



Algorithm

1. Initialize 10 *random* cluster means (one cluster mean for each digit 0-9)
   a. (K Means ++) Higher probability of being initialized further away from each other
2. For each handwritten digit, assign it to the closest cluster mean
3. For each of cluster C_i (i=1,...,10), calculate the mean of all the handwritten digits in cluster C_i
4. Repeat steps 2-3 until convergence
5. After convergence, classify each cluster to represent a digit by the closest digit mean

Flaws

- Finding local instead of global minima
- Initialization sensitivity

K Means

70.4668

K Mean

3.4556	3.9059	2.5862	4.2024	2.2800	2.2633	3.9923

K Means ++ % Accuracy:

72.4457	70.8755	73.4782	70.8970	73.0695	72.8544	72.9834

K Means ++ Run Time:

0.2210	0.1868	0.1661	0.1209	0.1464	0.1386	0.1109

K Means Code

step 1

```
%initialize cluster means
a = -1;
b = 1 ;
k=10;
n=256;
clusterInitMeans = a + (b-a).*rand(n,k)
oldMeans = clusterInitMeans;
tol=10^(-13);
endBool = false;
```

step 2

step 3

step 4

```
while endBool == false
    idx = []; %cluster assignment for each handwritten digit
    newMeans = zeros(256,10);
    for i=1:4649
        J = Inf;
        clusterIDX = nan;
        %for each handwritten digit, find the closest mean
        for j=1:10
            %euclidean distance
            new_J = norm(test_patterns(:,i) - oldMeans (:,j));
            if new_J < J
                J = new_J;
                clusterIDX = j;
            end
        end
        %classify handwritten digit
        idx(i) = clusterIDX;
    end

    %determine how many clusters there are (could be less than 10)
    clusterIDs = unique(idx);
    numclusters = length(clusterIDs);

    for i=1:numclusters %recalculate cluster means
        points = find(idx==clusterIDs(i));
        meanPoints = mean(test_patterns(:,points),2);
        newMeans(:,i) = meanPoints;
    end
    %change in cluster means changed to determine convergence
    if sum(abs(oldMeans - newMeans),"all") < tol
        endBool = true;
    else
        oldMeans = newMeans;
    end

end
```

step 5

```
C_256x10 = newMeans;

class_kmeans = NaN(10,1); % store classification should be 10x1
for i=1:10
    J = Inf; % initialize large distance between centroids
    label = nan;
    for j=1:10 % for each cluster-mean, find the closest training mean
        new_J = norm(C_256x10(:,i) - means(:,j));
        if new_J < J
            J = new_J;
            label = j-1;
        end
    end
    class_kmeans(i) = label;
end

%label index cluster for each data point as the corresponding digit
test_class_kmeans = NaN(1,4649);
for i=1:4649
    test_class_kmeans(i) = class_kmeans(idx(i));
end
```

K-Nearest-Neighbors Algorithm

- Let $P = [\overrightarrow{p_1}, \overrightarrow{p_2}, \dots, \overrightarrow{p_{4649}}]$ be our train matrix where $\overrightarrow{p_i} \in \mathbb{R}^{256}, 1 \leq i \leq 4649$
- Let $Q = [\overrightarrow{q_1}, \overrightarrow{q_2}, \dots, \overrightarrow{q_{4649}}]$ be our test matrix where $\overrightarrow{q_j} \in \mathbb{R}^{256}, 1 \leq j \leq 4649$

After finding d for each $\overrightarrow{p_i}$ from a given $\overrightarrow{q_j}$, Let D' be the subset of k, p_i vectors with lowest d.

** Equally weighted distance voting:

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D'} (v = y_i)$$

** Inverse-weighted distance voting:

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D'} w_i \times (v = y_i)$$

Weighting is commonly inverse wrt distance: w_i=1/d_i

Algorithm:
0: Pick small positive integer k
1: Find distance of input vector to all other vectors in dataset.
2: Extract k vectors with lowest distances
- If weighted: assign w=1/d to k nearest data points.
3: Assign class label to input vector:
- Unweighted: most common label out of the k vectors.
- Weighted: highest scoring y_i label associated with score: Sum((w_i) . (v = y_i)), out of the k nearest x_i vectors.

Flaws:
- Need to determine k, High computational cost, High storage, Suitable distance for the dataset should be used

Accuracy: 96.6%,
Run Time: 56.23s

```
% Equal weight, k=5, Euclidean distance:
    tic % runtime
    k=5;
    pred_labels = NaN(4649,1); % predicted labels of test vectors
    for i=1:4649 %iterate through test vectors
        train_vec_labels = NaN(4649,2); % col1:train labels, col2:distance from train to test
        for j=1:4649 %iterate through train vectors
            train_vec_labels(j,1) = train_label_vec(j); % train label
            train_vec_labels(j,2) = norm(train_patterns(:,j)-test_patterns(:,i)); % distance from test
        end
        neighbors = sortrows(train_vec_labels,2); % distance ascending order
        pred_labels(i) = mode(neighbors(1:k,1)); %select label mode of lowest 5 distances
    end
    toc % runtime
    pred_labels;

    %confusion matrix
    confusionMatrix = confusionmat(test_label_vec,pred_labels);
    %test)label_vec: 1x4649
    %pred_labels: 1x4649

    accuracy_percent = (sum(diag(confusionMatrix))/sum(confusionMatrix,"ALL"))*100;
```

Manhattan	Euclidean	Cubic	Cosine
$d = \sum_{i=1}^{n} \lvert p_i - q_i \rvert$	$d = \sqrt{\sum_{i=1}^{n} \lvert p_i - q_i \rvert^2}$	$d = \sqrt[3]{\sum_{i=1}^{n} \lvert p_i - q_i \rvert^3}$	$d = 1 - \frac{\vec{p}.\vec{q}}{\lVert\vec{p}\rVert.\lVert\vec{q}\rVert}$

Cosine Distance

- More robust than Minkowski distances for comparing vectors of highly differing magnitude (considers unit vectors).

Cosine Similarity:

$$s = \frac{\vec{p}.\vec{q}}{\lVert\vec{p}\rVert_2.\lVert\vec{q}\rVert_2}, s \in [-1,1]$$

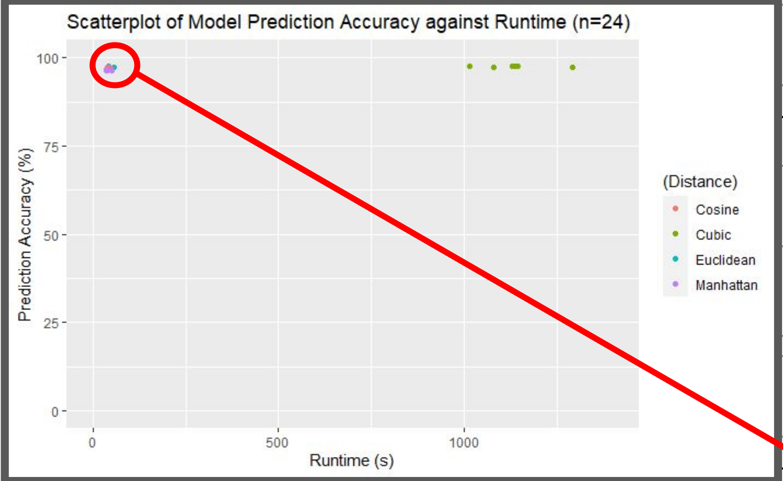
Cosine Distance

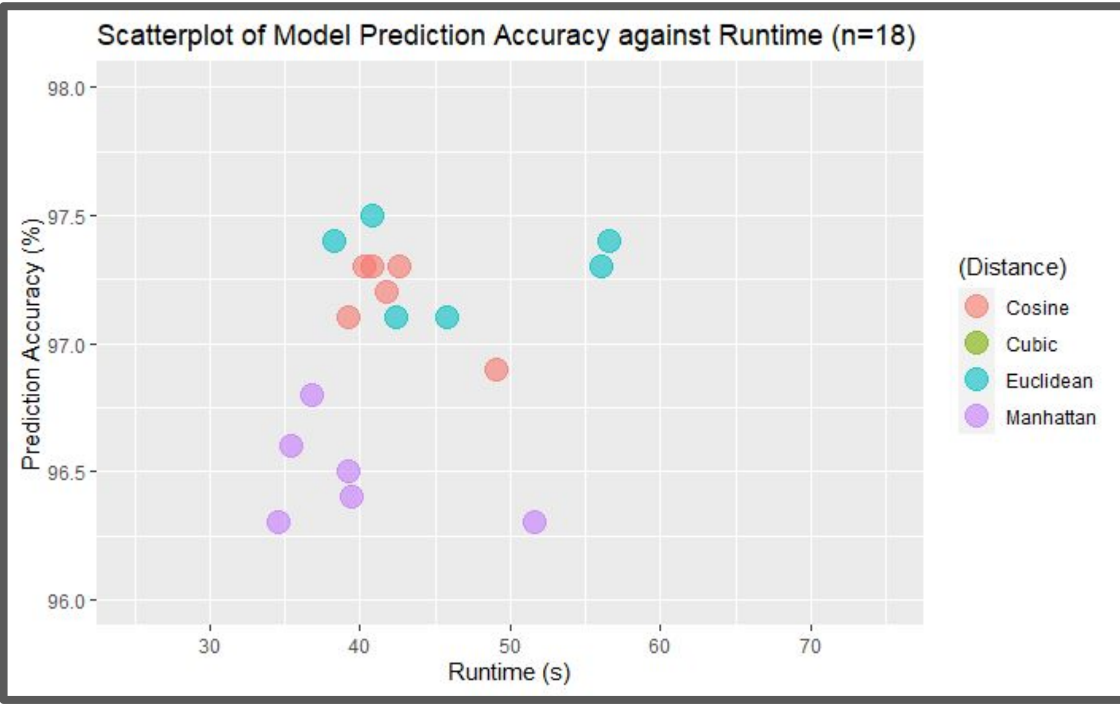
$$d = 1 - s, d \in [0,2]$$

s = 1: if p and q are parallel
s = 0: if p and q are orthogonal
s > 0: almost parallel
s < 0: almost antiparallel
s = -1: antiparallel

kNN Performance

These computations use 10-fold cross validation (10% testing, 90% training) with combinations: k=1,3,5, Inverse Weighted or Equally Weighted, Euclidean, Cosine, Cubic distance





k=1	k=3	k=5	k=1	k=3
97.7%, 1144.6s	97.5%, 1134.5s	97.4%, 1079.4s	97.6%, 1129.3s	97.4%, 1292.5s

Highest accuracy:

Model 1

True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10	TPR	FNR
1	99.5%		0.1%	0.1%		0.1%	0.1%	0.1%	0.1%		99.5%	0.5%
2		99.4%			0.4%		0.2%	0.1%			99.4%	0.6%
3	0.5%	0.2%	96.3%	0.5%	0.2%	0.3%	0.2%	0.8%	0.8%	0.1%	96.3%	3.7%
4	0.4%	0.1%	0.6%	96.5%		1.5%			0.7%	0.2%	96.5%	3.5%
5	0.2%	0.5%	0.2%		96.4%	0.1%	0.2%	0.2%	0.1%	2.0%	96.4%	3.6%
6	0.8%		0.7%	1.7%		95.4%	0.4%	0.1%	0.6%	0.3%	95.4%	4.6%
7	0.5%	0.1%	0.4%			0.4%	98.6%			0.1%	98.6%	1.4%
8		0.3%		0.1%	0.9%	0.1%		97.3%	0.4%	0.9%	97.3%	2.7%
9	0.7%	0.6%	0.1%	0.6%		1.0%		0.4%	95.9%	0.7%	95.9%	4.1%
10	0.1%				0.9%			0.1%	0.1%	98.8%	98.8%	1.2%

Lowest Run time:

Model 8

True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10	TPR	FNR
1	99.3%		0.2%	0.3%		0.1%		0.1%	0.1%		99.3%	0.7%
2		99.1%			0.6%		0.2%	0.1%			99.1%	0.9%
3	0.8%	0.2%	95.6%	0.8%	0.2%	0.2%	0.3%	1.2%	0.6%	0.1%	95.6%	4.4%
4	0.7%	0.1%	0.6%	95.6%		1.7%			1.0%	0.2%	95.6%	4.4%
5	0.2%	0.8%	0.2%	0.1%	92.7%	0.1%	0.5%	0.2%	0.1%	4.9%	92.7%	7.3%
6	1.7%		0.6%	3.2%	0.1%	91.6%	1.4%	0.3%	0.7%	0.4%	91.6%	8.4%
7	0.7%	0.2%	0.2%		0.1%	0.4%	98.3%				98.3%	1.7%
8		0.1%	0.1%		1.1%	0.1%		97.0%	0.3%	1.3%	97.0%	3.0%
9	1.7%	0.8%	0.4%	1.7%	0.3%	1.3%	0.1%	0.6%	92.4%	0.7%	92.4%	7.6%
10	0.1%			0.2%	1.3%			1.6%	0.2%	96.5%	96.5%	3.5%

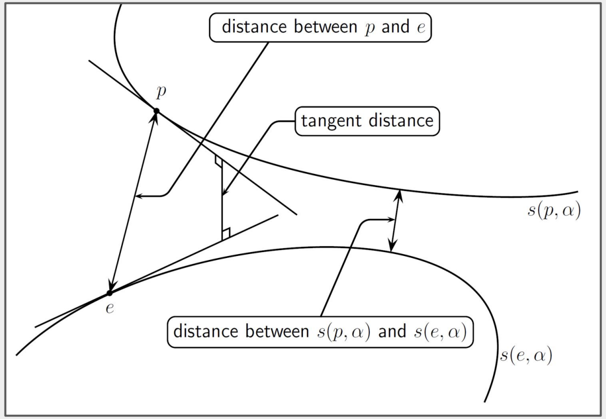
Tangent Distance Algorithm

Background

• s(p, α_p) → rotations of the digit p by a parameterized curve, where α_p is the angle of rotation

• transformation types: horizontal/vertical shift, rotation, scaling, thickness

• small rotations have minimal effect on tangent distance, unlike Euclidean distance



A Least Squares Problem

• to find distance between digits p and d, need distance s(p, α_p) and s(d, α_d)

• parametrized curves ≈ Taylor expansion

$$s(p,\alpha_p) = s(p,0) + \frac{ds}{d\alpha_p}(p,0)\alpha_p + \mathcal{O}(\alpha_p^2) \approx p + t_p\alpha_p$$

$$\text{where } t_p = \tfrac{ds}{d\alpha}(p,0)$$

• tangent distance = min distance between curves

$$\min_{\alpha_p,\alpha_d} \|s(p,\alpha_p) - s(d,\alpha_d)\|_2^2 \approx \min_{\alpha_p,\alpha_d} \|(p + t_p\alpha_p) - (d + t_d\alpha_d)\|_2^2$$

$$= \min_{\alpha_p,\alpha_d} \|(p-d) - (-t_p \quad t_d)(\alpha_p \quad \alpha_d)^T\|_2^2 = t_{pd}.$$

• α_p is only one transformation → what if more?

Tangent Distance Algorithm

A Least Squares Problem (continued)

- k transformations on p: $a_p = (\alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_k)^T$
- Multivariate Taylor expansion:

$$s(p, a_p) = s(p, 0) + \sum_{i=1}^{k} \frac{\partial s}{\partial \alpha_i}(p, 0)\alpha_i + \mathcal{O}(\|a_p\|_2^2) \approx p + T_p a_p$$

$$T_p = \begin{pmatrix} \frac{\partial s}{\partial \alpha_1} & \frac{\partial s}{\partial \alpha_2} & \dots & \frac{\partial s}{\partial \alpha_k} \end{pmatrix}$$

- Multivariate tangent distance:

$$\min_{a_p, a_d} \|(p + T_p a_p) - (d + T_d a_d)\|_2^2$$

$$= \min_{a_p, a_d} \|(p - d) - (-T_p \quad T_d)(a_p \quad a_d)^T\|_2^2 = t_{pd}.$$

- $T_p \rightarrow$ matrix of derivatives of transformations

$$T_p = \begin{pmatrix} \frac{\partial s}{\partial \alpha_1} & \frac{\partial s}{\partial \alpha_2} & \dots & \frac{\partial s}{\partial \alpha_k} \end{pmatrix}$$

Transformations

Let $f(x, y) \in \mathbb{R}$ be a differentiable function such that for a digit matrix $P \in \mathbb{R}^{16\times16}$, $f(i, j) = P_{ij}$ for all $i, j \in \{1, 2, ..., 16\}$ (e.g. $f(3, 4) = P_{3,4}$).

- derivatives of the transformations at α = 0:

Translation in the x direction	f_x
Translation in the y direction	f_y
Rotation about the "origin"	$yf_x - xf_y$
Scaling	$xf_x + yf_y$
Stretch/compress along the horizontal and vertical axes	$xf_x - yf_y$
Stretch/compress along the diagonals	$yf_x + xf_y$
Thickening	$(f_x)^2 + (f_y)^2$

- example for scaling: plug transformation in f(x,y), differentiate using chain rule

$$s(p, \alpha_s)(x, y) = f((1 + \alpha_s)x, (1 + \alpha_s)y)$$

$$\frac{d}{d\alpha_s}(s(p, \alpha_s)(x, y))|_{\alpha_s=0} = xf_x + yf_y.$$

Tangent Distance Algorithm

Algorithm

```
For each train/test image:
      Find f_x and f_y of image
Determine transformation function: T_x = f_x
```

```
For each test image, p:
      For each train image, d:
            Subtract train p from test d
            Compute tangent matrices and combine
            Solve least squares problem
            Get residual

      Find minimum residual among train images
      Set prediction to label of smallest residual
```

$$\min_{a_p, a_d} \|(p + T_p a_p) - (d + T_d a_d)\|_2^2$$

$$= \min_{a_p, a_d} \|(p - d) - (-T_p \quad T_d)(a_p \quad a_d)^T\|_2^2 = t_{pd}.$$

Code for Horizontal Transformation

```
p_x_train = []; p_y_train = [];
for i = 1:len_train
   image = reshape(train_data(:,i),[16,16])';
   [Gx,Gy] = imgradientxy(image);
   Gx_flat = reshape(Gx,[1,256]);
   Gy_flat = reshape(Gy,[1,256]);
   p_x_train = [p_x_train; Gx_flat];
   p_y_train = [p_y_train; Gy_flat];
```

```
predictions = [];
for t = 1:len_test
   residuals = [];
   for r = 1:len_train
       A = [-p_x_train(r,:); p_x_test(t,:)]';
       b = train_data(:,r) - test_data(:,t);
       [x,flag,relres] = lsqr(A, b);
       residuals = [residuals relres];
   [min_resid,ind] = min(residuals);
   disp(min_resid)
   predictions = [predictions train_labels(ind)]
```

Tangent Distance Algorithm

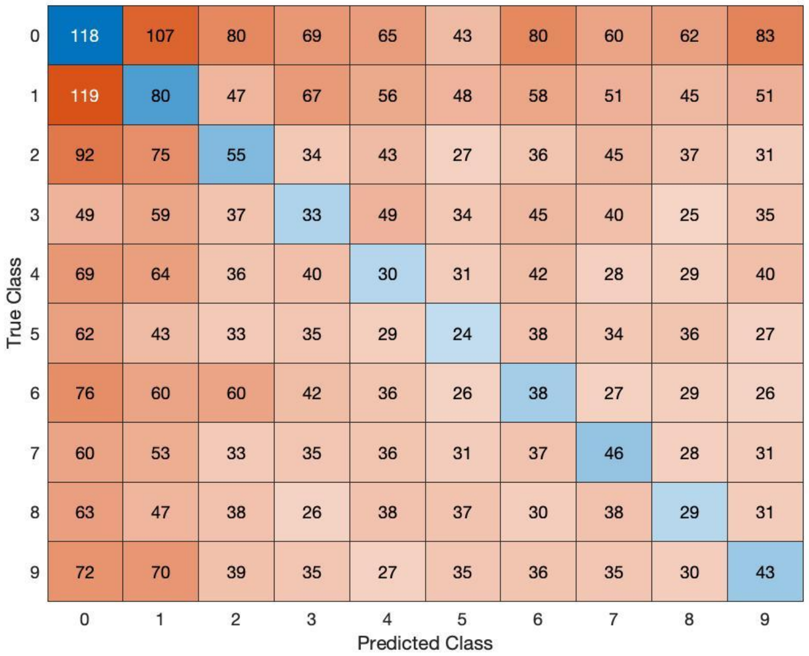
Results

USPS dataset + our implementation:
10.66%, not much better than random
2417 seconds = 40 minutes

MNIST + outside source:
21.35% without smoothing
91.41% with smoothing

Challenges

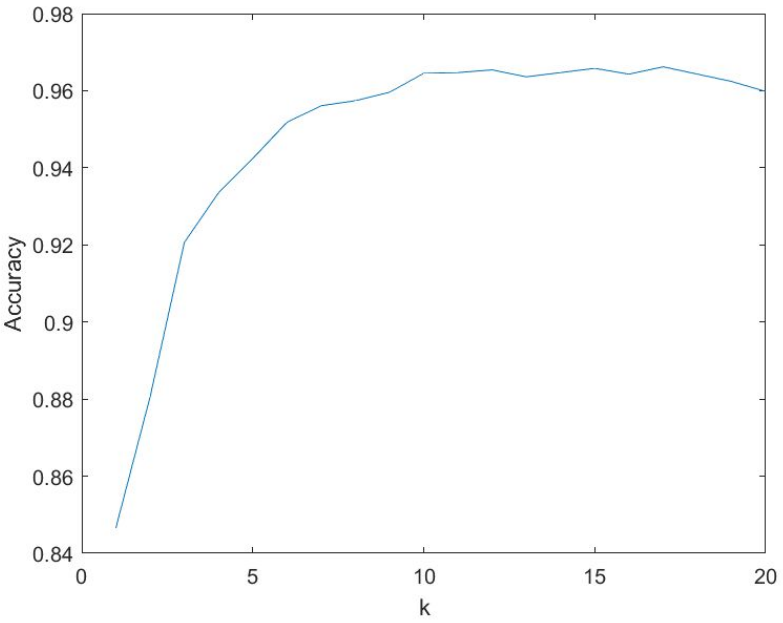
- works poorly without smoothing
- much slower and more expensive
- *length(test)* · *length(train)* # of comparisons
- does not significantly outperform other methods



SVD Classification Algorithm

- Data: $\vec{x}_1, \vec{x}_2, ...\vec{x}_{4649} \in \mathbb{R}^{256}$
- Take vectors representing the same digit, and put
  for 10 digits)
- Compute SVD for each digit matrix, and take $\vec{u_1}$
  range(DigitMatrix)
- Goal: To classify some handwritten digit vector,
  which digit matrix's first k left singular vectors (

$$||\vec{d}_i - U_k U_k^T \vec{d}_i||_2$$



MATLAB Code

```matlab
%%% Using k largest left singular vectors as approximate bases (Training)
k = 10
[U_0 S_0 V_0] = svds(zeros, k, 'largest')
[U_1 S_1 V_1] = svds(ones, k, "largest")
[U_2 S_2 V_2] = svds(twos, k, "largest")
[U_3 S_3 V_3] = svds(threes, k, 'largest')
[U_4 S_4 V_4] = svds(fours, k, 'largest')
[U_5 S_5 V_5] = svds(fives, k, 'largest')
[U_6 S_6 V_6] = svds(sixs, k, "largest")
[U_7 S_7 V_7] = svds(sevens, k, "largest")
[U_8 S_8 V_8] = svds(eights, k, 'largest')
[U_9 S_9 V_9] = svds(nines, k, 'largest')
```

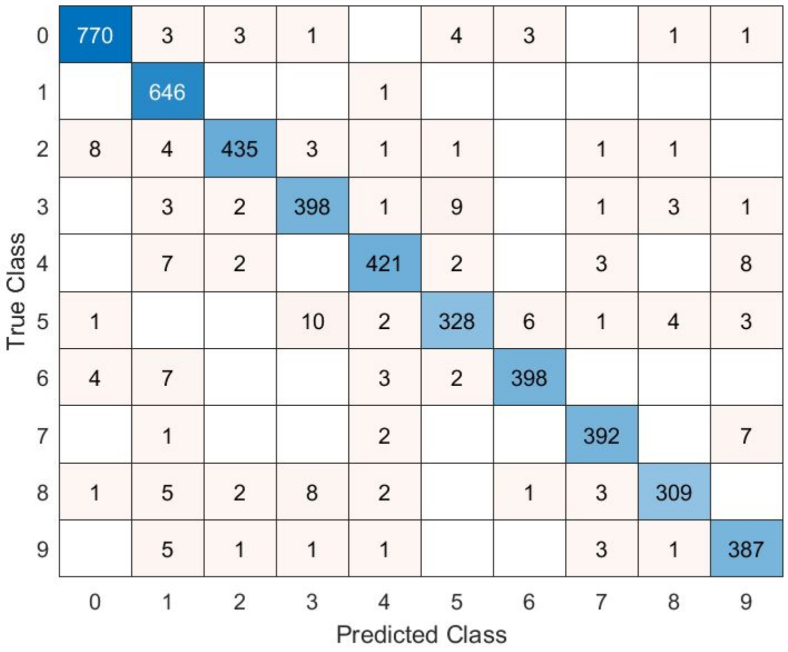
```matlab
predClass = []
for i=1:size(testMat, 2)
    v = testMat(:, i)
    res0 = norm(v-U_0*U_0'*v, 2)
    res1 = norm(v-U_1*U_1'*v, 2)
    res2 = norm(v-U_2*U_2'*v, 2)
    res3 = norm(v-U_3*U_3'*v, 2)
    res4 = norm(v-U_4*U_4'*v, 2)
    res5 = norm(v-U_5*U_5'*v, 2)
    res6 = norm(v-U_6*U_6'*v, 2)
    res7 = norm(v-U_7*U_7'*v, 2)
    res8 = norm(v-U_8*U_8'*v, 2)
    res9 = norm(v-U_9*U_9'*v, 2)

    residuals = [res0 res1 res2 res3 res4 res5 res6 res7 res8 res9]
```

```matlab
    if min(residuals) == res1
        classif = 1
    elseif min(residuals) == res0
        classif = 0
    elseif min(residuals) == res2
        classif = 2
    elseif min(residuals) == res3
        classif = 3
    elseif min(residuals) == res4
        classif = 4
    elseif min(residuals) == res5
        classif = 5
    elseif min(residuals) == res6
        classif = 6
    elseif min(residuals) == res7
        classif = 7
    elseif min(residuals) == res8
        classif = 8
    elseif min(residuals) == res9
        classif = 9
    end
    predClass(i) = classif
end
```

Results of SVD

Accuracy: 96.45%
Runtime: 219.452 s

- 3’s often misclassified as 5’s
- 9’s often misclassified as 4’s
- Overall, algorithm made some (reasonable) mistakes.



Algorithm Evaluation

Ranking	Algorithm	Accuracy (%)	Run Time (s)
3	Simple Classification	86.44	~ .15
4	K means ++	~72	~ 3
1	KNN (Manhattan, k=1, inv weight)	96.3	34.6
5	Tangent	10.66	2417
2	SVD	96.45	219.5

Digit	# Observations
0	767
1	622
2	475
3	406
4	409
5	355
6	420
7	390
8	377
9	422

Algorithm performance may be affected by the difference in observations per digit

Citations

- Saito's Notes: https://www.math.ucdavis.edu/~saito/courses/167.s17/Lecture21.pdf
- Textbook CH 10
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://en.wikipedia.org/wiki/Cosine_similarity
- https://bib.dbvis.de/uploadedFiles/155.pdf
- https://github.com/drCtul/3m201_groupe8/blob/23234790779fd95a10126758ff006caee949ed2e/tangent_distance_algorithm/tangent_distance_algorithm.ipynb
- https://stackoverflow.com/questions/13340353/distance-between-hyperplanes/13352507#13352507
- https://towardsdatascience.com/understanding-k-means-k-means-and-k-medoids-clustering-algorithms-ad9c9fbf47ca
- NMNV468: Numerical Linear Algebra for data science and informatics; Lecture 3: Handwriting Recognition and Classification
- Algorithms for Handwritten Digit Recognition; Michael J. M. Mazack, Western Washington University