Unsupervised Learning

(ML Assignment 3)

Silviu Pitis
GTID: spitis3
silviu.pitis@gmail.com

1 Datasets

As in Assignment 1, my motivation for picking my datasets was to work with baby problems in both computer vision and NLP. I was able to reuse the MNIST dataset for computer vision, but had to change datasets for NLP, since unsupervised learning did not make sense for sentiment analysis.

A MNIST: Handwritten Digits

As before, I use a subset of the full 70,000 MNIST images as the baby computer vision problem, with the following training, validation, test splits:

- **Training:** First 5000 samples from the base test set
- **Validation:** Last 5000 samples from the base test set
- **Test:** All 5000 samples from the base validation set.

All unsupervised learning / dimensionality reduction uses only the training set. Validation and test sets are used for rerunning the neural network learner.

I think MNIST remains a very interesting dataset for unsupervised learning because we can modify it, by provided only a limited number of labeled samples, to make it very close to the real-world "few label" semi-supervised learning task: in reality we see the same object from many different angles (this is like having a lot of unlabeled samples), but are only given its label once or twice until we generalize. The one difference is that in reality, the flow of time and invariance of the object we are viewing provides a simple method for clustering.

B Word Embeddings with Brothers Grimm

For the baby version of the NLP problem, I chose to try and create word2vec-like word embeddings[1] using only the clustering and dimensionality techniques we studied. Ideally, I wanted to see if we could (1) form clusters of similar words, and (2) observe the arithmetic relation "King – Queen + Woman = Man", for which word2vec is so well known.

For my corpus, I chose the "Fairy Tales" by Brothers Grimm,[2], which totals just over 100,000 words. This choice was motivated by trying to restrict the scope of the corpus to a small, well defined setting (fairy tales) that did not contain too many words (children's stories should have a restricted vocabulary). After limiting the vocabulary to include only words occurring at least 5 times, the total vocabulary was 1629 tokens.

Given the tokenized corpus, my dataset consisted of a word-word co-occurence matrix using context windows of size 11 (5 words to the left and 5 words to the right of the target word). For example, given a context size of 1, the corpus "the cat ate the cat", which has the vocabulary ["the", "cat", "ate"], divides into the context windows (target words in bold) ["**the** cat", "the **cat** ate", "cat **ate** the", "ate **the** cat", "the **cat**"], and produces the co-occurence matrix (with zeroed diagonal):

1 https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
2 https://www.gutenberg.org/files/2591/2591-h/2591-h.htm

```
[[0, 5, 3], // the
 [5, 0, 3], // cat
 [3, 3, 0]] // ate
```

Thus, my initial dataset had 1629 samples (one for each token), each represented by a vector of length 1629, where each component was the co-occurence count. As the unnormalized counts gave poor results, I normalized the vectors by dividing by the l2 norm (also tried normalizing with the l1 norm, which gave similar results to l2 normalization).

2 Clustering

A Summary of clustering methods

I clustered both datasets using k-means and a gaussian mixture model (which uses the EM algorithm under the hood).

K-means works by initializing k random “centers” (means) and repeatedly applying expectation (associating each point with the closest mean) and maximization (moving the means to best describe the clustered points) steps to convergence. Because convergence is not guaranteed to be optimal, we usually run k-means with several different initializations and pick the best result (which might be based on diversity, but in the case of scikit-learn, which I used, is based on minimizing the sum of distances of each point to the closest cluster center).

The gaussian mixture model uses the same algorithm, also with k clusters, except that (1) instead of hard cluster edges defined by midpoints between centers, each center defines a multi-dimensional gaussian model, (2) points are associated with the center that gives them the highest probability of being generated by the model defined by that center, and (3) during the maximization step, we not only allow the centers to move, but also allow their (co)variance matrices to change.

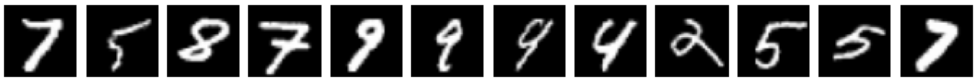
Both k-means and gaussian mixtures depend critically on the measure of distance (or conversely, similarity) between examples. By default, we use euclidean distance between the vector representations. This is addressed for each dataset separately below.

B MNIST

MNIST has ten digits, and so the natural “k” to use is 10. Using a smaller “k” does not really make sense, but I did try higher ks (20 and 50) to see if that would further disambiguate clusters (e.g., some 4s are closed and some are not, and a higher “k” may give each type of 4 its own cluster).

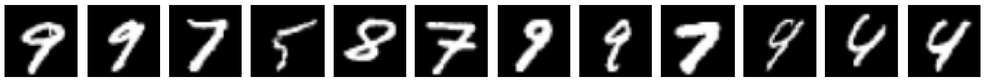
As MNIST has labels provided, the best way I could think to evaluate the quality of the clusters is to see how “pure” they are in a single digit. Thus, I evaluated each cluster based on what percent of that cluster consisting of the most common digit in that cluster. When using k=10, k-means achieved an overall purity of about 60% (60.5% with 10 random initializations, and 61.6% with 100 random initializations). All digits but 5 were accounted for---that is, the purest digit in each of the 10 clusters was, respectively: 0, 1, 2, 3, 4, 6, 6, 7, 8, 9. Interestingly, only about 8% of the “6” clusters consisted of 5s, whereas 22% of the 8 cluster and 24% of the 3 cluster consisted of 5s.

As expected, increasing k resulted in progressively better purity, with k=20 giving an overall purity of 70% and k=50 giving an overall purity of about 80% (but note that if k = len(dataset), the purity with be 100%). Nevertheless, even with higher Ks, the least pure cluster was particularly poor. For example, one of the k=20 clusters had a purity of only 22%. Here are 12 samples from this cluster:



Given the diversity of these samples, it is difficult to see just what they have in common (but we’ll come back to this when discussing PCA) that would cause k-means to produce this result.

Interestingly, this same cluster is produced upon rerunning k-means, and a very similar cluster is produced upon running the mixture of gaussian model with k=20:



On the whole, the mixture of gaussians produced very similar results to k-means, as long as the covariance of the means was fully defined (i.e., the covariance matrix was not restricted to a diagonal matrix). These results are summarized below:

	K-means Normalized	K-means K = 10	K-means K = 20	K-means K = 50	GMM Diag covar	GMM K = 10	GMM K = 20	GMM K = 50
Purity:	52%	61%	70%	79%	44%	61%	70%	69%

Note that the first result column above is “K-means normalized”, which is the result when each feature is first scaled to have zero mean and unit variance. Basically, I found that scaling the features implied a poorer measure of distance / similarity between digits than leaving them in their original form. We can reach this same conclusion by thinking about what scaling does: by scaling the variance, we are essentially reducing the distance contribution of features that vary often, as compared to those that do not. But this doesn’t really make sense. All in all, I could not think of a better definition of distance than that induced by the original representation / euclidean distance.

C Word Embeddings

The purpose of clustering the word embeddings is to see if can obtain semantically or syntactically meaningful clusters of similar words. Unlike MNIST, there are no labels for the data, and so it is difficult to find an objective measure of cluster quality. Therefore, my assessment of word embedding clusters is highly subjective, based on my own background knowledge of the English language. This is not to say that clustering is pointless---in might find similarities I would not have thought of in the absence of clustering, and in any case, automatic clustering of equal quality to manual clustering would clearly be useful.

Whereas scaling was harmful to the MNIST clustering results, using raw (unnormalized) co-occurence vectors leads to very poor clusters. This is because the most common words have very large feature vectors, due to the high co-occurence counts of those words. Thus, entire clusters are formed around the most common words (one word per cluster). This is fixed, by first normalizing each co-occurence vector with either its l1 or l2 norm.

Here are three example clusters for different normalizations:

	K-means Unnormalized, K=10	K-means L1 normalized, K=10	K-means L2 normalized, K=10
Cluster 1	<unk>	`great, gave, such, while, fine, lived, bread, wine, piece, large, stone, kind, fellow, fat, rich, strange, short, pig, handsome, …, sword, deal, changed, bush, …`	`great, long, beautiful, such, while, fine, lived, wine, piece, large, stone, kind, clever, fellow, fat, strange, short, pig, whose, handsome, …, sword, deal, changed, wonderful…`
Cluster 2	`the`	`and, of, in, then, when, on, at, into, came, little, king, s, by, man, old, …, woman, golden, horse, brought, poor, fox, cat, …, queen, young, boy, behind, fire, evening, prince, …`	`king, water, princess, morning, forest, next, wood, castle, wolf, under, world, side, window, peasant, third, soldier, sun, youth, garden, …`
Cluster 3	a	`door, tree, bird, water, princess, forest, wolf, tailor, under, world, dwarf, cook, window, peasant, third, soldier, sun, … youth, garden, …`	`began, going, work, wanted, run, drink, sent, given, carry, forced, length, ordered, luck, agreed, able, pass, begged, …`
...	...	...	...

At a glance, while we don't get very nice categories like "nouns", "royalty", etc., the clusters do seem to capture some underlying structure. It is hardly obvious what that structure is, but, e.g., the 1st clusters shown in the l1 and l2 normalized columns seem to have a lot of adjectives, whereas the 3rd cluster shown in the l2 normalized column contains predominantly verbs. The normalization (distance measure) clearly has an effect, but again, it is not obvious how to characterize this effect, or if one of the two clusterings is preferable.

I tried increasing the number of clusters to 50 and 100 in order to try and get better defined clusters, but this only marginally helped. While using K=100 does end up grouping "beautiful", "fine", "rich", "handsome", "splendid", "mighty", and "wonderful" in the same cluster, that very same cluster contains the words "screaming", "snake", and "draught". There is a pattern, but it falls just short of something that is truly nice and meaningful.

The clusters produced by gaussian mixtures were very similar to those produced by K-means. By only eyeballing the differences I could not determine a significant qualitative difference.

3 Dimensionality Reduction

A Summary of techniques

For this part, I applied PCA, ICA, random projections and a sparse, discrete autoencoder to the data.

PCA uses SVD to find the basis vectors corresponding to the top K singular values of the data matrix, and produces a linear transformation that does a change of basis from the elementary basis vectors in the original features space to the space defined by the PCA basis vectors. This linear transformation is then used to transform the data into the new space. The idea is that the top singular values correspond to the orthogonal axes along which the variance of the data is greatest.

ICA, motivated by the blind source separation problem, attempts to factor the original data into a linear combination of K independent sources. The original idea was to maximize the independence of the source reconstructions by maximizing some notion of their "non-gaussianity" (e.g., their kurtosis, differential entropy, or the negative of their mutual information), whilst simultaneously maximizing the information they hold about the original data. A more modern understanding, which uses the same underlying algorithm, models each source as an independent non-gaussian probability distribution (usually, the distribution defined by the a sigmoidal CDF), and learns a transformation in order to maximize the likelihood that the original data would be produced. Unlike the original understanding, which is non-constructive with respect to the source distributions, the modern understanding is assumptive with respect to the sources and can be used generatively.

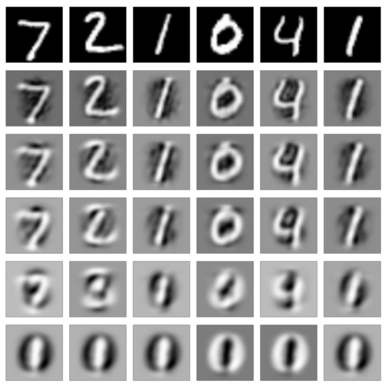
Random projections, rather than solving any analytic problem (like PCA), or optimization problem (like ICA), simply apply a random linear transformation to the data. I don't know that there is any theory that says one kind of random matrix is better than another, but I just used scikit-learn's default of a Gaussian random projection, where the projection matrix draws each component from N(0, 1/K).

The sparse, discrete autoencoder (DAE) is of my own design, and consists of a single hidden layer with K one-hot neurons. I used 3-bit neurons, so that each neuron is represented by an 8 dimensional one-hot vector, and K neurons form an embedding of total size 8*K. The embedding is learned by minimizing the L2 reconstruction error of the autoencoder, using the straight-through estimator in order to train the one-hot neurons.[3]

3 I used a similar autoencoder in my blog post (http://r2rt.com/deconstruction-with-discrete-embeddings.html), and I describe the one-hot straight through estimator in more detail in another blog post (http://r2rt.com/beyond-binary-ternary-and-one-hot-neurons.html)

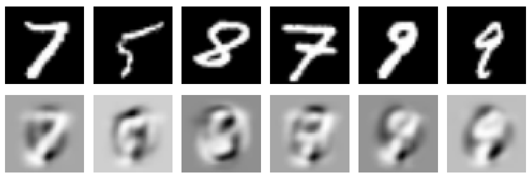
B MNIST

If we apply PCA to MNIST, we obtain the following results:

Dimensions	Reconstructions	Percent of Variance Explained	Reconstruction Error
Original		100%	0
100 dims		92%	0.045
50 dims		82%	0.051
30 dims		73%	0.058
10 dims		48%	0.073
1 dim		10%	0.097

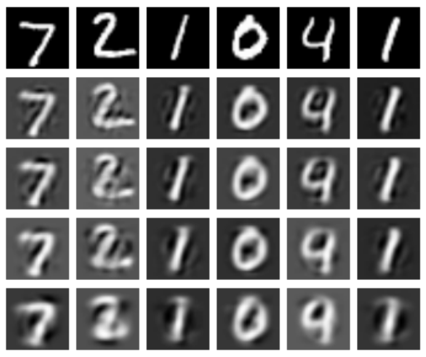
The above gives us a good idea of the kind of compression that we can obtain using PCA. At more than 50 dimensions, the transformed data still contains over 80% of the original information and produces reconstructions that are very identifiable with the originals. Below 50 dimensions, the reconstructions become poorer. All semblance of the original is lost when it is reduced to a single dimension (as expected).

We can use PCA to understand the mystery cluster found by K-means above (the one with very poor purity). Observe the reconstructions of samples from that cluster after they have been reduced to 20 dimensions using PCA:



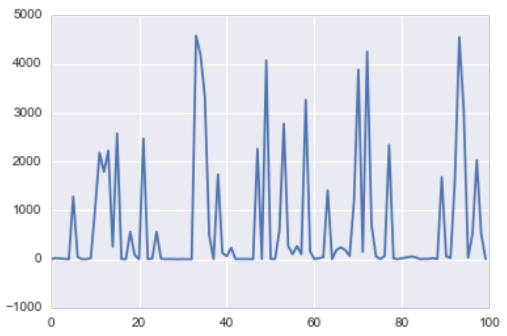
Note how there is a very distinct black area in the bottom right, a distinctly white area to the bottom left, and a small black area at the top. It is likely that all three of these contribute to the grouping of these distinct digits into a single cluster by K-means.

Applying ICA, we obtain the following:

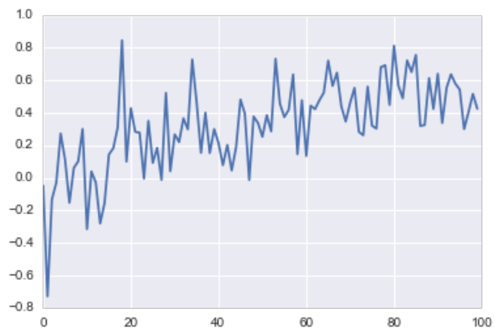
Dimensions	Reconstructions	Reconstruction Error
Original		0
100 dims		0.012
50 dims		0.019
30 dims		0.024
10 dims		0.036

Note that while I did not scale MNIST prior to applying PCA (it obtained worse results), I did scale it prior to applying ICA, in order to be consistent with theory. Overall ICA produces lower reconstruction error than PCA (not too surprising, given that ICA is optimizing for this). At the same time, the reconstructions created by ICA are distinctly "fuzzier", in the sense that they become progressively more blurry, as opposed to becoming progressively more deformed like PCA.

By examining the kurtosis of the ICA components, we can confirm that the algorithm is working correctly. Since the objective is to find non-gaussian sources, their (absolute) kurtosis should ideally be very high (maximal) (since gaussian sources would have 0 kurtosis). If we compare the kurtosis of the components (using 100 dimensions) returned by PCA and ICA, we see that there is a very real difference, which confirms that ICA is doing what it purports to be doing:

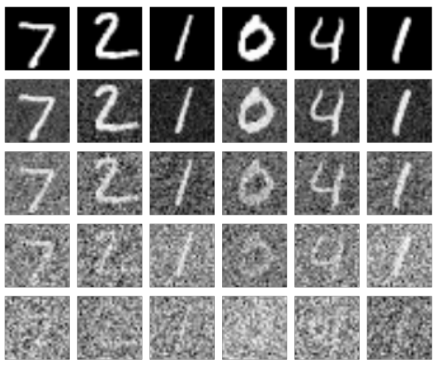


ICA Kurtosis (absolute value)



PCA Kurtosis (absolute value)

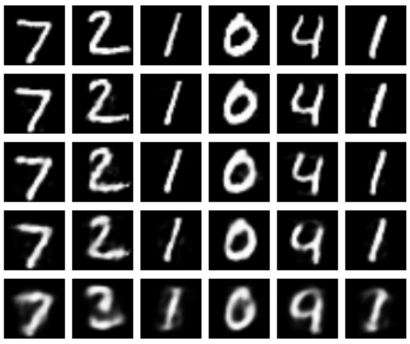
Randomized projections produce the below results:

Dimensions	**Reconstructions**	**Reconstruction Error**
Original		0
700 dims		0.011
400 dims		0.051
150 dims		0.084
50 dims		0.097

Note that I had to leave the number of dimensions much higher than PCA/ICA to get reasonable reconstructions. Here, and in general, the reconstructions are formed using the pseudo-inverse of the NxM transformation matrix.

Random projections produce much noisier reconstructions than PCA and ICA, with much larger reconstruction error. Interestingly, running the random projection multiple times produces very similar results, both in terms of visual appearance and in terms of reconstruction error. From this, we can conclude that different sets of random basis vectors end up capturing about the same amount of information.

The discrete autoencoder is optimized so as to minimize reconstruction error, and so it is not surprising that it outperforms all three of the other methods in terms of reconstruction. Note that although the raw number of dimensions is much larger than the other methods, each neuron has only 8 values can be represented in just 3 bits, which actually makes this representation more than 10x more compressed than any of the others (where each dimension is a 32-bit floating point number):

Dimensions	**Reconstructions**	**Reconstruction Error**
Original		0
100 3-bit neurons (800 binary dims)		0.005
50 3-bit neurons (400 binary dims)		0.010
30 3-bit neurons (240 binary dims)		0.016
10 3-bit neurons (80 binary dims)		0.036

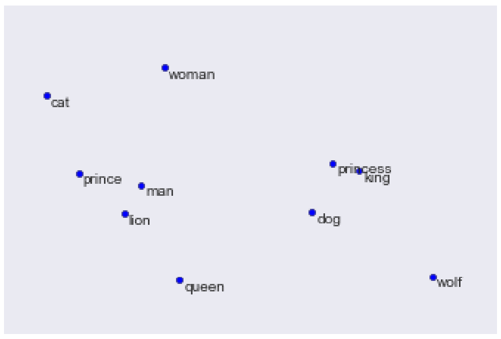
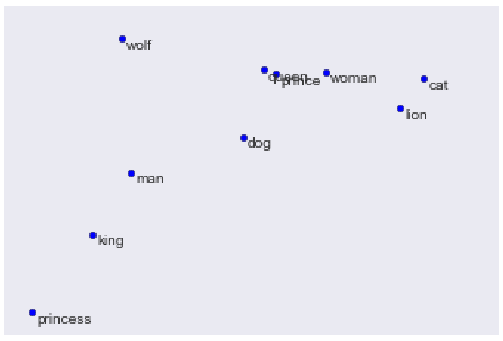
C *Word Embeddings*

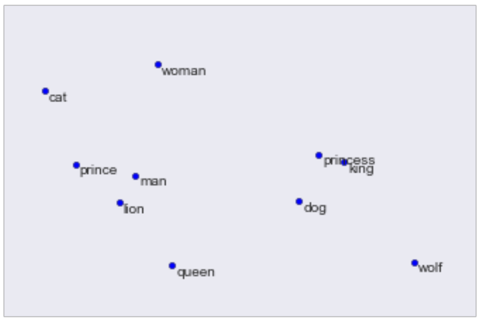
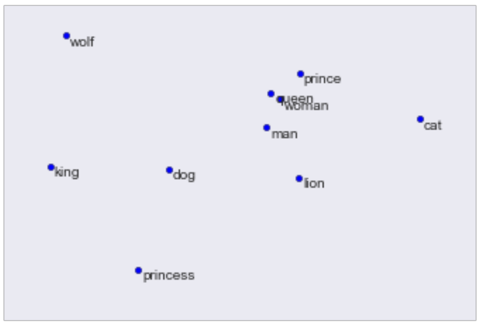
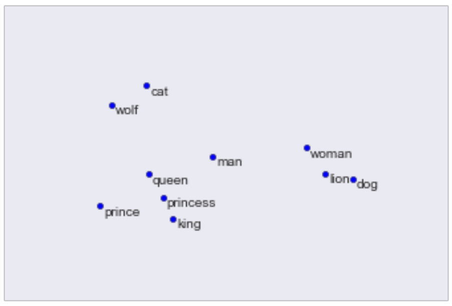
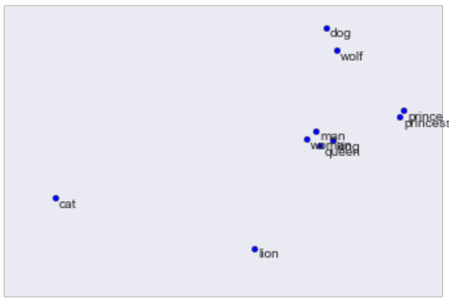
I next tried to see if dimensionality reduction can create semantically meaningful word embeddings. The hope is that similar words will be closer together in the embedded space. Here are the reconstruction errors for each of the four dimensionality reduction methods:

	PCA	ICA	RP	DAE
100 components	`0.00044`	`0.00014`	`0.00057`	`0.00024`
50 components	`0.00046`	`0.00017`	`0.00059`	`0.00022`
30 components	`0.00048`	`0.00018`	`0.00060`	`0.00023`
10 components	`0.00052`	`0.00020`	`0.00061`	`0.00024`

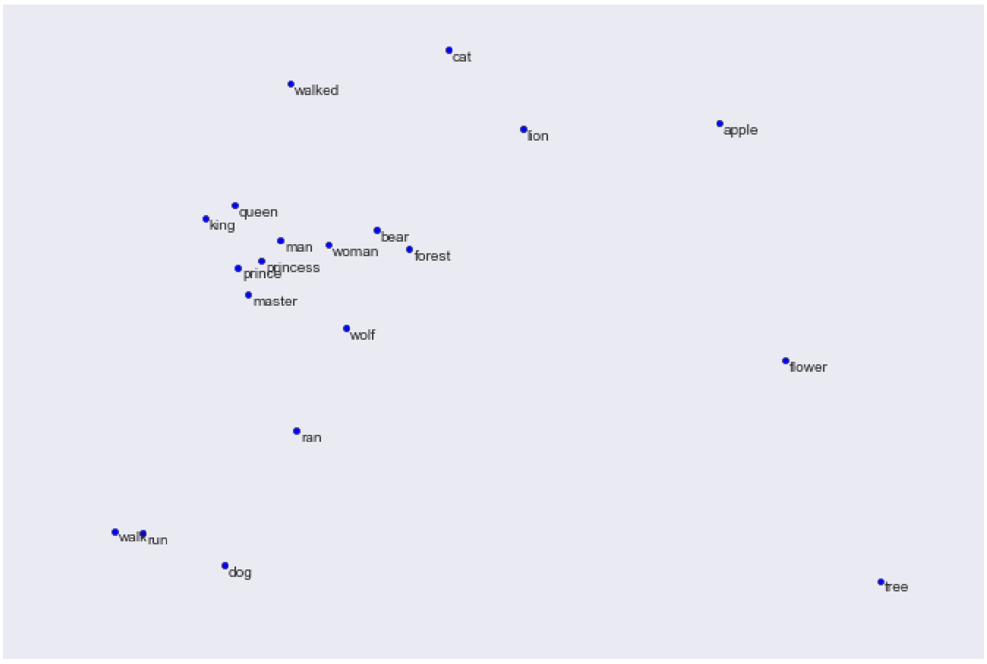
As you can see, a similar pattern of reconstruction error arises (except the DAE does worse than ICA, perhaps because it uses hyperparameters that were tuned to autoencoding MNIST), even though the context is different, which suggests that reconstruction is very closely linked to the dimensionality reduction algorithm.

Unlike MNIST, where reconstruction was one of our key objectives, here we are more interested in finding meaning in the word co-occurence vectors. To see if dimensionality reduction is meaningful, I further reduce the dimensionality to 2, using both PCA and TSNE methods, to see if we can spot a meaningful arrangement of certain keywords. The results are shown below.

	PCA	**TSNE**
Original (l2-normalized) vectors	 *No (consistently) meaningful relationships*	 *No (consistently) meaningful relationships*

PCA (K = 50)	 *No (consistently) meaningful relationships*	 *No (consistently) meaningful relationships*
ICA (K = 50)	 *No (consistently) meaningful relationships*	 *Similar words are close to each other!*

The results for RP and DAE showed no meaningful relationships. What was interesting, however, was that ICA together with a TSNE projection found exactly what I wanted. Here is a projection with more (20) keywords and K=30:



Unlike all of the other projections, there are no “nonsensical” relationships. All the persons are together, with the king/queen, man/woman, prince/princess pairs all closest to each other. We also observe that walk/run are close, and that ran/walked (past tense) are in the same relative direction.

This is a nice result that strongly suggests that the ICA-reduced word vectors live on a semantically-meaningful manifold. Query how well these word vectors would perform on supervised tasks when compared to the vectors produced by word2vec.

4 Dimensionality Reduction → Clustering

In this part, I ran the clustering algorithms after reducing the dimensionality of the data as in Part 3. I ran the clustering experiments with the dimensions reduced to 100 and to 30. For each dataset, I used the same criteria as above to determine the quality of the clusters (purity for MNIST, subjective quality judgment for the word embeddings).

MNIST - When using PCA, the results, with both 100 dimensions and 30 dimensions, were very much the same as with the original data. The same observations hold: there was no cluster that was most pure in 5, as the 5s got confused with other digits. The accuracy was the same as with the original data. All other forms of dimensionality reduction tended to decrease the purity of the MNIST clusters, with DAE performing somewhat better than ICA and RP. The DAE's poorer performance may be explained by the fact that it results in discrete embeddings, and euclidean distance in discrete space (as used by K-means and GMM) seems like it would be suboptimal (but I offer no better alternatives). ICA's poor performance can be explained by MNIST not meeting the assumptions of ICA (it is not clear what the independent sources causing MNIST would be – perhaps penstrokes could serve as sources, but then they would most definitely not be independent). Finally, RP's poor performance can be explained by the fact that it is random, and there is no reason to expect it to perform well.

Embeddings – As before, other than "eyeballing" the quality, I don't have a very good method to judge cluster quality for word embeddings. For most methods, the clusters looked the same as above.

Given the good results of ICA in the previous part, I paid particular attention to see if it would result in clearly better clusters. I think it did (K-means with K=10, using features from ICA with M=30):

Food/drink-related: some, eat, drink, bread, cow, wine, piece, ate, blow, fetch, eaten, food, pretty, grandmother, frederick, fat, draw, cellar, peas, drank, bad, salad, pig, order, catherine, plenty, draught, cheese, cake, beer, pot, meat, hungry, cup, certainly, ale, thirsty, hat, curdken, sword, milk, hurt, taking, plate, nuts, new, dry, steak, roast …	**Common words / pronouns:** <unk>, ,, the, and, to, he, a, was, it, she, in, her, his, but, him, they, had, as, so, with, then, when, at, all, came, went, one, little, there, them, king, s, were, this, who, by, away, again, their, very, man, time, old, day, saw, father, home, before, himself, about …	**Prepositions:** of, on, into, out, up, down, took, upon, off, from, which, over, door, put, fell, sat, tree, ran, bird, head, water, gave, golden, horse, herself, red, gold, white, lay, cut, hand, behind, fast, fire, under, laid, eyes, flew, side, dead, run, open, hair, fine, rose, full, sun, right, its, …
Horse/carriage: chanticleer, nor, partlet, horses, neither, dirty, wheels, six, fundevogel, carriage, lina, ride, spinning, built, mice, coach, harnessed …	**Dialogue?** you, said, that, i, for, not, will, !, be, have, is, me, ?, what, my, if, no, do, go, would, could, your, now, are, did, good, we, come, how, should, well, take, shall, must, let, …	**Random (numbers?):** lie, days, juniper, kywitt, paid, thousand, few, bones, between, deal, bought, yellow, underneath, nut, five, eight, goods, buttons, lap, treasure, share, miles, dig, bury, ago…

Several of the clusters have very clear semantic themes. This result is clearly far better than a random clustering, and the fact that we were able to obtain these word groupings from a relatively small corpus and with such relatively fast / straightforward techniques speaks volumes.

Whereas ICA was not very good at modeling MNIST, it was the best technique for modeling the word embeddings. This makes sense in terms of the assumptions of ICA. As noted above, it is difficult to see how handwritten digits can be caused by independent sources, but it is easy to see how independent sources can make up the contexts surrounding words---for example, words just tell a story, and inside a story the scene, people in the scene, animals inside the scene, weather, etc. are all more or less independent sources of meaning (or if not independent, they can be modeled that way to greater success than MNIST).

5 Dimensionality reduction → NN; Clustering → NN

Only the MNIST dataset had labels, and so I restricted my attention for this part to MNIST. I used each technique to reduce the dimensionality of MNIST from 784 features to 100 features, and then trained a two layer network with a hidden layer of 1000 neurons to classify the data. In order to test the quality of features produced by clustering, I the clustering algorithms in order to create a one-hot feature. Because a single one-hot feature could perform no better than its purity, I created several such one hot features by using different values of K. For example, I created 4 one-hot features using K=16, K=32, K=64, and K=128, which produced a feature vector with a total of 240 binary dimensions, where only 4 dimensions were "on".

In each case, I used early stopping on the validation set, and recorded the final accuracy on the test set. Because the neural network training was practically free (I ran experiments on a GPU), there was no significant different in the time required to train.

The results on the test set are summarized in the following table:

	Baseline from P1	PCA M = 100	ICA M = 100	RP M = 100	DAE M = 100	Kmeans K=[16, 32, 64, 128]	Kmeans K=[2, 4, 8, 16, 32, 64, 128]	Kmeans K=[8, 16, 32]	GMM K=[8, 16, 32]
Loss	**0.21**	0.27	0.37	0.30	0.38	0.59	0.59	0.79	1.25
Acc	**95%**	94%	92%	92%	92%	85%	84%	76%	61%
Time	3.13	2.08	3.19	2.62	2.29	2.04	2.12	2.17	10.5

No change in features was able to beat the baseline 784 features. This is not particularly surprising since this is a simple dataset, neural networks are powerful, and dimensionality reduction techniques can really only throw out potentially useful information in this case. For more complex datasets, I could see some of information being thrown out by dimensionality reduction or clustering being noisy information, so that generalization error would go down due to reduced overfitting. Here, that does not seem to be the case.

Of the change in features, all dimensionality reduction techniques showed respectable performance that was significantly greater than the performance of the clustering techniques. This is not surprising because the number of dimensions maintained by the dimensionality reduction techniques was effectively 20+ times greater than the number of dimensions that resulted from clustering. I would note that features formed by clustering are very closely related to those features that would be obtained by KNN. With 5000 datapoints and K = 128, we are doing something very close to KNN with K = 5000/128 = 39 nearest neighbors (instead of using neighbors, we are using landmarks that represent the centers of clusters of neighbors). For this reason, it is not surprising that the features formed by clustering did pretty good, as KNN was one of the most effective techniques from project 1, achieving an accuracy of about 92%.

Of the dimensionality reduction techniques, PCA performed best. This was somewhat surprising to me because the reconstructions that result from using the DAE, and even ICA, are far better than the ones resulting from the use of PCA. For this reason, I think there is also an element of learn-ability / inductive bias that changes when the features are changed, but I am not sure what.

Perhaps surprisingly, a random projection with K=100 (whose reconstruction looks almost like pure noise) did very well on the classification task. I think this is more a testament to the power of neural networks than the power of random projections, as the information that was left in the projected digits was sufficient for classification.

Finally, I wanted to comment on the fact that gaussian mixtures performed more poorly than Kmeans, both here, and in the clustering section. I'm not really sure why this happened, as they are supposed to be a more expressive model. I would wager that this is merely a consequence of the fact that the assumptions of the model do not fit the data very well (much like how the assumptions of ICA do not fit the MNIST data very well).