k-Nearest Neighbors by Means of Sequence to Sequence Deep Neural Networks and Memory Networks

Yiming Xu , **Diego Klabjan**
Northwestern University
x.yiming@outlook.com, d-klabjan@northwestern.edu

Abstract

k-Nearest Neighbors is one of the most fundamental but effective classification models. In this paper, we propose two families of models built on a sequence to sequence model and a memory network model to mimic the k-Nearest Neighbors model, which generate a sequence of labels, a sequence of out-of-sample feature vectors and a final label for classification, and thus they could also function as oversamplers. We also propose 'out-of-core' versions of our models which assume that only a small portion of data can be loaded into memory. Computational experiments show that our models on structured datasets outperform k-Nearest Neighbors, a feed-forward neural network, XGBoost, lightGBM, random forest and a memory network, due to the fact that our models must produce additional output and not just the label. On image and text datasets, the performance of our model is close to many state-of-the-art deep models. As an oversampler on imbalanced datasets, the sequence to sequence kNN model often outperforms Synthetic Minority Over-sampling Technique and Adaptive Synthetic Sampling.

1 Introduction

Recently, neural networks have been attracting a lot of attention among researchers in both academia and industry, due to their astounding performance in fields such as natural language processing and image recognition. Interpretability of these models, however, has always been an issue since it is difficult to understand the performance of neural networks. The well-known manifold hypothesis states that real-world high dimensional data (such as images) form lower-dimensional manifolds embedded in the high-dimensional space [Carlsson *et al.*, 2008], but these manifolds are tangled together and are difficult to separate. The classification process is then equivalent to stretching, squishing and separating the tangled manifolds apart. However, these operations pose a challenge: it is quite implausible that only affine transformations followed by pointwise nonlinear activations are sufficient to project or embed data into representative manifolds that are easily separable by class.

Therefore, instead of asking neural networks to separate the manifolds by a hyperplane or a surface, it is more reasonable to require points of the same manifold to be closer than points of other manifolds [Olah, 2014]. Namely, the distance between manifolds of different classes should be large and the distance between manifolds of the same class should be small. This distance property is behind the concept of k-Nearest Neighbor (kNN) [Cover and Hart, 1967]. Consequently, letting neural networks mimic kNN would combine the notion of manifolds with the desired distance property.

We explore kNN through two deep neural network models: sequence to sequence deep neural networks [Sutskever *et al.*, 2014] and memory networks [Weston *et al.*, 2015][Sukhbaatar *et al.*, 2015]. A family of our models are based on a sequence to sequence network. The new sequence to sequence model has the input sequence of length 1 corresponding to a sample, and then it decodes it to predict two sequences of output, which are the classes of closest samples and neighboring samples not necessarily in the training data, where we call the latter as out-of-sample feature vectors. We also propose a family of models built on a memory network, which has a memory that can be read and written to and is composed of a subset of training samples, with the goal of using it for predicting both classes of close samples and out-of-sample feature vectors. With the help of attention over memory vectors, our new memory network model generates the predicted label sequence and out-of-sample feature vectors. Both families of models use loss functions that mimic kNN. Computational experiments show that the new sequence to sequence kNN model consistently outperforms benchmarks (kNN[Cover and Hart, 1967], random forest[Breiman, 2001], XGBoost[Chen and Guestrin, 2016], lightGBM[Ke *et al.*, 2017], a feed-forward neural network and a vanilla memory network) on structured datasets. The performance on some commonly used image and text datasets is comparable to many state-of-the-art deep models. We postulate that this is due to the fact that we are forcing the model to 'work harder' than necessary (producing out-of-sample feature vectors).[1]

Different from general classification models, our models predict not only labels, but also out-of-sample feature vectors. Usually a classification model only predicts labels, but

[1]Please refer to http://dynresmanagement.com/uploads/3/5/2/7/35274584/knn_ijcai_appendix.pdf for the appendix of our paper.

as in the case of kNN, it is desirable to learn or predict the feature vectors of neighbors as well. Intuitively, if a deep neural network predicts both labels and feature vectors, it is forced to learn and capture representative information of input, and thus it should perform better in classification. Moreover, our models also function as synthetic oversamplers: we add the out-of-sample feature vectors and their labels (synthetic samples) to the training set. Experiments show that our sequence to sequence kNN model outperforms Synthetic Minority Over-sampling Technique (SMOTE) [Chawla *et al.*, 2002] and Adaptive Synthetic sampling (ADASYN) [He *et al.*, 2008] most of the times on imbalanced datasets.

Usually we allow models to perform kNN searching on the entire dataset, which we call the full versions of models, but kNN is computationally expensive on large datasets. We design an algorithm to resolve this and we test our models under such an 'out-of-core' setting: only a batch of data can be loaded into memory, i.e. kNN searching in the entire dataset is not allowed. For each such random batch, we compute the K closest samples with respect to the given training sample. We repeat this R times and find the closest K samples among these KR samples. These closest K samples provide the approximate label sequence and feature vector sequence to the training sample based on the kNN algorithm. Computational experiments show that sequence to sequence kNN models and memory network kNN models significantly outperform the kNN benchmark under the out-of-core setting.

Our main contributions are as follows. First, we develop two types of deep neural network models which mimic the kNN structure. Second, our models are able to predict both labels of closest samples and out-of-sample feature vectors at the same time: they are both classification models and oversamplers. Third, we establish the out-of-core version of models in the situation where not all data can be read into computer memory or kNN cannot be run on the entire dataset. The full version of the sequence to sequence kNN models and the out-of-core version of both sequence to sequence kNN models and memory network kNN models outperform the benchmarks, which we postulate is because learning neighboring samples enables the model to capture representative features.

2 Background and Literature Review

There are several works trying to mimic kNN or applying kNN within different models. Based on the boundary forest model, in [Zoran *et al.*, 2017], a boundary deep learning tree model with differentiable loss function was presented to learn an efficient representation for kNN. The main differences between this work and our work are in the base models used (boundary tree vs standard kNN), in the main objectives (representation learning vs classification and oversampling) and in the loss functions (KL divergence vs KL divergence components reflecting the kNN strategy and L^2 norm). [Wang *et al.*, 2017] introduced a text classification model which utilized nearest neighbors of input text as the external memory to predict the class of input text. Our memory network kNN models differ from this model in 1) the external memory: our memory network kNN models simply feed a random batch of samples into the external memory without the requirement of nearest neighbors and thus they save computational time and 2) they considered only a classification setting, while our models generate not only labels but also out-of-sample feature vectors. Most importantly, the loss functions are different: in [Wang *et al.*, 2017] the authors used KL divergence as the loss function while we use a specially designed KL divergence and L^2 norm to force our models to mimic kNN.

The sequence to sequence model, one of our base models, has recently become the leading framework in natural language processing [Sutskever *et al.*, 2014][Cho *et al.*, 2014]. In [Cho *et al.*, 2014] an RNN encoder-decoder architecture was used to deal with statistical machine translation problems. In [Sutskever *et al.*, 2014] the authors proposed a general end-to-end sequence to sequence framework. The major difference between our work and these studies is that the loss function in our work forces the model to learn from neighboring samples, and our models are more than just classifiers - they also create out-of-sample feature vectors that improve accuracy or can be used as oversamplers.

There are also a plethora of studies utilizing external memory in neural networks. [Weston *et al.*, 2015] proposed the memory network model to predict the correct answer of a query by means of ranking the importance of sentences in the external memory. [Sukhbaatar *et al.*, 2015] introduced a continuous version of a memory network with a recurrent attention mechanism over an external memory, which outperformed the previous discrete memory network architecture in question answering.

In summary, the main differences between our work and previous studies are as follows. First, our models predict both labels of nearest samples and out-of-sample feature vectors rather than simply labels. Thus, they are more than classifiers: the predicted label sequences and feature vector sequences can be treated as synthetic oversamples to handle imbalanced class problems. Second, our work emphasizes on the out-of-core setting. All of the prior works related to kNN and deep learning assume that kNN can be run on the entire dataset and thus cannot be used on large datasets. Third, our loss functions are designed to mimic kNN, so that our models are forced to learn neighboring samples to capture the representative information.

3 kNN Models

Our sequence to sequence kNN models are built on a Seq2seq model, and our memory network kNN models are built on a MemN2N model. The details of Seq2seq model and memory network model can be found in the appendix. Let K denote the number of neighbors of interest.

3.1 Vector to Label Sequence (V2LS) Model

Given an input feature vector x, a ground truth label Y^{GT} (a single class corresponding to x) and a sequence of labels $Y_1^T, Y_2^T, ..., Y_K^T$ corresponding to the labels of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest sample to x in the entire training set, V2LS predicts a label Y^P and $Y_1^P, Y_2^P, ..., Y_K^P$, the predicted labels of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest samples. Since $Y_1^T, Y_2^T, ..., Y_K^T$ are obtained by using kNN upfront, the real input is only x and Y^{GT}. When kNN does not misclassify, Y^{GT} corresponds to majority voting of $Y_1^T, Y_2^T, ..., Y_K^T$.

The key concept of our model is to have x as the input sequence (of length 1) and the output sequence $Y_1^P, Y_2^P, ..., Y_K^P$ to correspond to $Y_1^T, Y_2^T, ..., Y_K^T$. The loss function also captures Y^{GT} and $Y_1^T, Y_2^T, ..., Y_K^T$.

In the V2LS model, by a softmax operation with temperature after a linear mapping (W_y, b_y), the label of the t^{th} nearest sample to x is predicted by $Y_t^P = softmax((W_y y_t + b_y)/\tau)$, where y^t is as in (1) in the appendix for $t = 1, 2, ..., K$ and τ is the temperature of softmax.

By taking the average of predicted label distributions, the label of x is predicted by $Y^P = \sum_{t=1}^K Y_t^P / K$. Note that if Y_t^P corresponds to a Dirac distribution for each t, then Y^P matches majority voting. Temperature τ controls the "peakedness" of Y_t^P. Values of τ below 1 push Y_t^P towards a Dirac distribution, which is desired in order to mimic kNN [Karpathy, 2015][Hinton *et al.*, 2015].

We design the loss function as $L_1 = E\{\sum_{t=1}^K D_{KL}(Y_t^T || Y_t^P)/K + \alpha D_{KL}(Y^{GT} || Y^P)\}$, where the first term captures the label at the neighbor level, the second term for the actual ground truth and α is a hyperparameter to balance the two terms. The expectation is taken over all training samples, and D_{KL} denotes the Kullback-Leibler divergence. Due to the fact that the first term is the sum of KL divergence between predicted labels of nearest neighbors and target labels of nearest neighbors, it forces the model to learn information about the neighborhood. The second term considers the actual ground truth label: a classification model should minimize the KL divergence between the predicted label (average of K distributions) and the ground truth label. By combining the two terms, the model is forced to not only learn the classes of the final label but also the labels of nearest neighbors. We let the t^{th} decoder cell predict the t^{th} nearest sample because the preceding decoder cells preserve closeness to the original input. In the subsequent cells, the information gets passed through more decoder cells and thus it is expected to deviate more from the input, which is why we let the t^{th} cell predict the t^{th} nearest neighbor.

In inference, given an input x, V2LS predicts Y^P and $Y_1^P, Y_2^P, ..., Y_K^P$, but only Y^P is the actual output; it is used to measure the classification performance. Note that it is possible that $argmaxY^P$ is different from the majority voted class among $argmaxY_1^P, argmaxY_2^P, ..., argmaxY_K^P$ when kNN misclassifies.

3.2 Vector to Vector Sequence (V2VS) Model

We use the same structure as the V2LS model except that in this model, the inputs are a feature vector x and a sequence of feature vectors $X_1^T, X_2^T, ..., X_K^T$ corresponding to the $1^{st}, 2^{nd}, ..., K^{th}$ nearest sample to x among the entire training set (calculated upfront using kNN). V2VS predicts $X_1^P, X_2^P, ..., X_K^P$ which denote the predicted out-of-sample feature vectors of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest sample. Since $X_1^T, X_2^T, ..., X_K^T$ are obtained using kNN, this is an unsupervised model.

The output of the t^{th} decoder cell y_t is processed by a linear layer (W_{x1}, b_{x1}), a $ReLU$ operation and another linear layer (W_{x2}, b_{x2}) to predict the out-of-sample feature vector $X_t^P = W_{x2} max\{W_{x1} y_t + b_{x1}, 0\} + b_{x2}, t = 1, 2, ..., K$. Numerical experiments show that $ReLU$ works best compared with $tanh$ and other activation functions.

The loss function is defined to be the sum of L^2 norms: $L_2 = E\{\sum_{t=1}^K ||X_t^P - X_t^T||^2\}$. Since the predicted out-of-sample feature vectors should be close to the input vector, learning nearest vectors forces the model to learn a sequence of approximations to something very close to the identity function. However, this is not trivial. First it does not learn an exact identity function, since the output is a sequence of nearest neighbors to input, i.e. it does not simply copy the input K times. Second, by limiting the number of hidden units of the neural network, the model is forced to capture the most representative and condensed information of input. A large amount of studies have shown this to be beneficial to classification problems [Erhan *et al.*, 2010][He *et al.*, 2016].

In inference, we predict the label of x by finding the labels of out-of-sample feature vectors X_t^P and then perform majority voting among these K labels. The most voted label is regarded as the prediction of the current sample.

3.3 Vector to Vector Sequence and Label Sequence (V2VSLS) Model

In previous models, V2LS learns to predict labels of nearest neighbors and V2VS learns to predict feature vectors of nearest neighbors. Combining V2LS and V2VS together, this model predicts both X_t^P and Y_t^P. Given an input feature vector x, a ground truth label Y^{GT}, a sequence of nearest labels $Y_1^T, Y_2^T, ..., Y_K^T$ and a sequence of nearest feature vectors $X_1^T, X_2^T, ..., X_K^T$, V2VSLS predicts a label Y^P, a label sequence $Y_1^P, Y_2^P, ..., Y_K^P$ and an out-of-sample feature vector sequence $X_1^P, X_2^P, ..., X_K^P$. Since the two target sequences are obtained by kNN, the model still only needs x and Y^{GT} as input.

The loss function is a weighted sum of the two loss functions in V2LS and V2VS: $L = L_1 + \lambda L_2$, where λ is a hyperparameter to account for the scale of the L^2 norm and the KL divergence.

The L^2 norm part enables the model to learn neighboring vectors. As discussed in the V2VS model, this is beneficial to classification since it drives the model to capture representative information of input and nearest neighbors. The KL part of the loss function focuses on predicting labels of nearest neighbors. As discussed in the V2LS model, the two terms in the KL loss force the model to learn both neighboring labels and the ground truth label. Combining the two parts, the V2VSLS model is able to predict nearest labels and out-of-sample feature vectors, as well as one final label for classification. The model is structured in this way because the K^{th} decoder cell output corresponds to the K^{th} nearest neighbor, so that the model is forced to better mimic kNN.

3.4 Memory Network - kNN (MNkNN) Model

The MNkNN model is built on the MemN2N model, which has K layers stacked together. After these layers, the

MemN2N model generates a prediction. In order to mimic kNN, our MNkNN model has K layers as well but it generates one label after each layer, i.e. after the t^{th} layer, it predicts the label of the t^{th} nearest sample. Similar to the Seq2seq kNN models, the t^{th} layer predicts the t^{th} nearest sample because the preceding layers preserve closeness to the input. Therefore, we let the preceding layers predict the closest nearest neighbors to mimic kNN.

This model takes a feature vector x, its ground truth label Y^{GT}, a random subset $x_1, x_2, ..., x_n$ from the training set (to be stored in the external memory) and $Y_1^T, Y_2^T, ..., Y_K^T$ denoting the labels of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest samples to x among the entire training set (calculated upfront using kNN). It predicts a label Y^P and a sequence of K labels of closest samples $Y_1^P, Y_2^P, ..., Y_K^P$.

After the t^{th} layer, by a softmax operation with temperature after a linear mapping (W_y, b_y), the model predicts the label of t^{th} nearest sample by $Y_t^P = softmax((W_y(Hu^t + o^t) + b_y)/\tau)$ where $t = 1, 2, ..., K$. The final label of x is calculated by $Y^P = \sum_{t=1}^{K} Y_t^P / K$.

The loss function of MNkNN is: $\overline{L_1} = E\{\sum_{t=1}^{K} KL(Y_t^T || Y_t^P)/K + \alpha KL(Y^{GT} || Y^P)\}$ which is the same as in V2LS. The first term accounts for learning neighboring information, and the second term forces the model to provide the best single candidate class.

In inference, the model takes a query x and random samples $x_1, x_2, ..., x_n$ from the training set, and generates the predicted label Y^P as well as a sequence of nearest labels $Y_1^P, Y_2^P, ..., Y_K^P$.

3.5 Memory Network - kNN with Vector Sequence (MNkNN_VEC) Model

This model is built on MNkNN, but it predicts out-of-sample feature vectors X_t^P as well. MNkNN_VEC takes a query feature vector x, its corresponding ground truth label Y^{GT}, a random subset $x_1, x_2, ..., x_n$ from the training dataset (to be stored in the external memory), $Y_1^T, Y_2^T, ..., Y_K^T$ and $X_1^T, X_2^T, ..., X_K^T$ denoting labels and feature vectors of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest samples to x among the entire training set (calculated both upfront using kNN). MNkNN_VEC predicts a label Y^P, a sequence of labels $Y_1^P, Y_2^P, ..., Y_K^P$ and a sequence of out-of-sample feature vectors $X_1^P, X_2^P, ..., X_K^P$.

By a linear mapping T, a $ReLU$ operation and another linear mapping (W_x, b_x), the feature vectors are then calculated by $X_t^P = W_x max\{T(Hu^t + o^t), 0\} + b_x$.

Same as the V2VSLS model, combining the L^2 norm and the KL divergence together, the loss function is defined as $\overline{L} = \overline{L_1} + \lambda E\{\sum_{t=1}^{K} ||X_t^P - X_t^T||^2\}$.

3.6 Out-of-Core Models

In the models exhibited so far, we assume that kNN can be run on the entire dataset exactly to compute the K nearest feature vectors and corresponding labels to an input sample. However, there are two problems with this assumption. First, this can be very computationally expensive if the dataset size is large. Second, the training dataset might be too big to fit in memory. When either of these two challenges is present, an out-of-core model assuming it is infeasible to run a 'full' kNN on the entire dataset has to be invoked. The out-of-core models avoid running kNN on the entire dataset, and thus save computational time and resources.

Let B be the maximum number of samples that can be stored in memory, where $B > K$. For a training sample x, we sample a subset S from the training set (including x) where $|S| = B$, then we run kNN on S to obtain the K nearest feature vectors and corresponding labels to x, which are denoted as $Y^T(S) = \{Y_1^T, Y_2^T, ..., Y_K^T\}$ and $X^T(S) = \{X_1^T, X_2^T, ..., X_K^T\}$ for x in the training process. The previously introduced loss functions L and $\overline{L}$ depend on $x, Y^{GT}, X^T(S), Y^T(S)$ and the model parameters Θ, and thus our out-of-core models are to solve

$$\min_{\Theta} E_x\, E_S\{\widetilde{L}(x, Y^{GT}, X^T(S), Y^T(S), \Theta)\}$$

where $\widetilde{L}$ is either L or $\overline{L}$.

Sampling a set of size B and then finding the nearest K samples only once are insufficient on imbalanced datasets, due to the low selection probability for minor classes. To resolve this, we iteratively take R random batches: each time a random batch is taken, we update the closest samples $X^T(S)$ by the K closest samples among the current batch and the K previous closest samples. These resulting nearest feature vectors and corresponding labels are used in the loss function. Note that we allow the previously selected samples to be selected in later sampling iterations. The entire algorithm is exhibited in Algorithm 1 in the appendix.

4 Computational Experiments

In this section, we evaluate our models on 9 classification datasets: Network Intrusion (NI) [Hettich and Bay, 1999], Forest Covertype (COV) [Blackard and Dean, 1998], SensIT [Duarte and Hu, 2004], Credit Card Default (CCD) [Yeh and hui Lien, 2009], MNIST[LeCun and Cortes, 1999], CIFAR-10[Krizhevsky *et al.*, 2009], News20[Lang, 1995], IMDb[Maas *et al.*, 2011] and Reuters [Chollet, 2015], which are all publicly available. Among the 9 datasets, the first 4 are structured and the remaining are unstructured.

For each dataset we experiment with 5 different seeds and all reported numbers are averages taken over 5 random seeds. We discuss the performance of the models in two aspects: classification and oversampling.

4.1 Classification

Experimental Setup

As comparisons against memory network kNN models and sequence to sequence kNN models, we use kNN with Euclidean metric and several currently best classification models random forest (RF), extreme gradient boosting (XGB), lightGBM (LGBM), a 4-layer feed-forward neural network (FFN) trained using the Adam optimization algorithm (which has been calibrated) with dropout and batch normalization and MemN2N (since MNkNN and MNkNN_VEC are built on MemN2N) as benchmarks. Value $K = 5$ is used in all models because it yields the best performance with low standard deviation among $K = 1, 2, ..., 20$. Increasing K beyond $K = 5$ is somewhat detrimental to the F-1 scores while significantly increasing the training time.

In the sequence to sequence kNN models, LSTM cells are used. In the memory network kNN models, the size of the external memory is 64 since we observe that models with memory vectors of size 64 generally provide the best F-1 scores with acceptable running time. Both sequence to sequence kNN models and memory network kNN models are trained using the Adam optimization algorithm with initial learning rate set to be 0.01. We also find that $\tau = 0.85$, $\lambda = 0.12$ and $\alpha = 9.5$ provide overall the best F-1 scores.

We first experiment on structured datasets not requiring special embeddings, i.e. NI, COV, SensIT and CCD. We only consider 3 classes in NI and COV datasets due to significant class imbalance.

Overall Results of Full Model on Structured Data

We first discuss the full models that can handle all of the training data, i.e. kNN can be run on the entire dataset. Table 1 shows that in the full model case, V2VSLS consistently outperforms the best classification models on all four datasets. t-tests show that it significantly outperforms benchmarks at the 5% level on all four datasets. For our kNN models, V2LS significantly outperforms V2VS, because V2VS tries to reconstruct the feature level information as explained in Section 3.2, which does not utilize the label information. Moreover, it can also be seen that predicting not only labels but feature vectors as well is reasonable, since V2VSLS consistently outperforms V2LS and MNkNN_VEC consistently outperforms MNkNN. Models predicting feature vectors outperform models not predicting feature vectors on all datasets. These memory based models exhibit subpar performance, which is expected since they only consider 64 training samples at once (despite using exact labels).

Overall Results of Full Model on Unstructured Data

To provide insights of how our model performs on unstructured data, we further evaluate on the image and text datasets: MNIST, CIFAR-10, News20, IMDb and Reuters. The embeddings are fed into classifiers.

We compare V2VSLS with some of the most popular classification models in Table 1. On News20, 7-layer FFN performs slightly better than V2VSLS, but V2VSLS consistently outperforms other classification models on all other datasets. There is a slight gap attributed to the single stage employed by pure deep learning models v.s. our experiment that has two stages (embedding construction, kNN). Nevertheless, the performance of V2VSLS on these unstructured datasets still outperforms many currently popular models.

Comparison with a Set-Based Model and Swapped Order V2VSLS

We evaluate the necessity of modeling nearest neighbors as a sequence, instead of as a set. First, we compare the set-based model with the V2VSLS model. Note that compared to V2VSLS, the set-based neural network model also predicts K labels, K nearest neighbors and a final label for classification, but it does not model the K labels and nearest neighbors as a sequence. The only difference is that the set-based model's outputs are orderless. As shown in Table 1, the set-based model which removes the order of nearest neighbors suffers from a consistent performance drop across datasets. The set-based model still outperforms most of the existing popular classification methods, however, which again validates that predicting nearest neighbors is beneficial for classification.

Following the orderless nearest neighbors experiment, we arbitrarily swap the first and the third nearest neighbor of the order in the training data. Intuitively, if the performance drops after swapping the nearest neighbors in the training data, utilizing the order information of nearest neighbors is crucial. The results are shown in Table 1. V2VSLS with swapped order performs worse than V2VSLS with original order, but it still outperforms the set-based model consistently, which validates that keeping the order of the nearest neighbors is necessary.

Please note that we have also compared our model with other related benchmark models, and the details can be found in the appendix.

Overall Results of Out-of-Core Model

Next, we validate our models in the out-of-core scenario which avoids running kNN on the entire dataset for saving computational resources. In the out-of-core versions of our models, R is set to be 50, since we observe that increasing 50 only has a slight impact on F-1 scores. However, this substantially increases the running time.

	NI	COV	SensIT	CCD
kNN	73.87	63.87	61.40	59.41
V2LS	90.63	90.29	82.47	67.51
V2VS	81.92	71.29	69.12	61.36
V2VSLS	**91.27**	**92.89**	**83.38**	**69.21**
MNkNN	81.89	78.58	78.80	66.16
MNkNN_VEC	83.19	81.72	82.32	68.15

Table 2: F-1 score of out-of-core model with R=50.

Table 2 shows the results of our models under the out-of-core assumption when $R = 50$ and $B = 64$. Both V2VLSL and MNkNN_VEC significantly outperform the kNN benchmark based on t-tests at the 5% significance level. The kNN benchmark provides a low score since we restrict the batch size (or memory size) to be 64, and it turns out that kNN is substantially affected by the randomness of batches. Our models are robust under the out-of-core setting, because the weight of the ground truth label in the loss function is relatively high so that even if the input nearest sequences are noisy, they still can focus on learning the ground truth label and making reasonable predictions.

Full Model and Out-of-Core Model Comparison

Table 3 shows a comparison between the full and out-of-core models with $R = 50, B = 64$ on the SensIT dataset. The running time of our models are broken down to two parts: the first part is the time to obtain sequences of K nearest feature vectors and labels and the second part is the model training time. Under the out-of-core setting, overall the kNN sequence preprocessing time is saved by approximately 40% while the models perform only slightly worse.

	NI	**COV**	**SensIT**	**CCD**	**MNIST**	**CIFAR-10**	**News20**	**IMDb**	**Reuters**
kNN	90.54	91.15	82.56	63.81	98.91	93.12	62.14	86.25	72.34
RF	90.44	93.76	82.70	66.94	98.87	92.69	58.55	87.41	73.70
XGB	87.53	91.98	82.56	66.95	99.12	91.55	69.81	88.51	74.24
LGBM	90.23	89.85	83.29	65.68	99.57	92.18	70.59	88.60	74.98
SVM	89.28	90.59	83.15	66.01	98.86	92.95	71.84	87.75	73.97
FFN	88.53	91.83	83.67	65.37	99.51	94.41	**72.93**	88.33	74.83
MemN2N	79.36	77.98	75.17	61.83	96.20	90.13	54.20	81.01	69.87
LambdaRank	45.58	59.81	41.03	38.91	70.18	62.59	49.71	65.97	41.69
kNN-AN	64.18	69.64	54.29	52.18	89.59	81.72	55.01	67.80	59.81
V2LS	91.28	93.94	84.93	68.38	99.51	94.18	72.39	87.71	75.77
V2VS	86.18	90.39	74.84	64.23	98.17	92.98	70.11	86.27	72.10
V2VSLS	**92.07**	**94.97**	**86.24**	**69.87**	**99.70**	**94.86**	72.68	**89.83**	**76.11**
MNkNN	83.83	80.12	79.58	67.26	94.38	89.10	62.33	84.18	69.28
MNkNN_VEC	84.59	83.94	83.41	68.82	97.29	93.02	71.54	87.85	74.17
Set-based model	91.25	94.10	85.51	68.77	99.51	93.39	70.88	88.19	74.91
Swapped V2VSLS	91.79	94.56	85.99	69.42	99.43	94.01	72.17	89.51	75.70

Table 1: F-1 score comparison of full models.

	kNN	**V2LS**	**V2VS**	**V2VSLS**	**MNkNN**	**MNkNN_VEC**
Full F-1	82.56	84.93	74.84	86.24	79.58	83.41
OOC F-1	61.40	82.47	69.12	83.38	78.80	82.32
Full time (a)	312	443	857	1391	443	1391
OOC time (a)	193	287	488	741	287	741
Full time (b)	NA	635	1358	1802	692	1081
OOC time (b)	NA	619	1316	1846	703	1055

Table 3: Full model and out-of-core (OOC) model comparison on SensIT. Time metrics are in seconds. (a) denotes the time to obtain sequences of nearest neighbors and (b) denotes the model training time.

	NI	**COV**	**SensIT**	**CCD**
FFN-original	89.64	91.83	83.67	65.37
FFN-SMOTE	89.99	91.18	83.43	66.32
FFN-ADASYN	90.38	90.67	83.72	66.51
FFN-V2VSLS	**90.89**	**92.05**	**83.94**	**66.82**
XGB-original	87.53	91.98	82.56	66.95
XGB-SMOTE	87.79	91.86	82.87	66.56
XGB-ADASYN	**88.39**	**92.56**	**83.42**	66.20
XGB-V2VSLS	87.62	92.43	82.46	**66.96**
RF-original	90.44	93.76	82.70	66.94
RF-SMOTE	89.97	93.88	83.01	66.13
RF-ADASYN	89.39	93.83	**83.34**	67.14
RF-V2VSLS	**90.79**	**94.36**	82.75	**68.08**

Table 4: Oversampling: F-1 score comparison.

4.2 Oversampling

Experimental Setup

When class distributions are highly imbalanced, many classification models have low accuracy or F-1 score on the minority class. A simple but effective way to handle this problem is to oversample the minority class. Since V2VSLS and MNkNN_VEC are able to predict out-of-sample feature vectors, we also regard our models as oversamplers and we compare them with two widely used oversampling techniques: SMOTE and ADASYN. We only test V2VSLS since it scales better than MNkNN_VEC and the prediction performance is comparable. In our experiments, we evaluate our model on four imbalanced datasets. We first fully train the model, and then for each sample from the training set, V2VSLS predicts $K = 5$ out-of-sample feature vectors which are regarded as synthetic samples. We add them to the training set if they are in a minority class until the classes are balanced or there are no minority training data left for creating synthetic samples. In our oversampling experiments, we use $\lambda = 1.3$ and $\alpha = 3$.

Overall Results on Oversampling

Table 4 shows the F-1 scores of FFN, extreme gradient boosting and random forest classification models, with different oversampling techniques, namely, original training set without oversampling, SMOTE, ADASYN and V2VSLS. V2VSLS performs the best among all combinations of classification models and oversampling techniques, as shown in Table 5. Although most of the time models on datasets with three oversampling techniques outperform models on datasets without oversampling, the classification performance still largely depends on the classification model used and which dataset is considered. The visualization of the synthetic samples can be found in the appendix.

	NI	**COV**	**SensIT**	**CCD**
Best model	FFN+V2VSLS	RF+V2VSLS	FFN+V2VSLS	RF+V2VSLS
Best F-1	90.89	94.36	83.92	68.08
Imp on SMOTE*	0.51%	0.61%	2.28%	1%
Imp on ADASYN*	0.6%	0.56%	0.26%	1.4%

Table 5: Oversampling techniques comparison. ‘Imp on X*’ denotes how much better than best performance of X.

5 Conclusion

In summary, we find that it is beneficial to have models learn not only labels but also feature vectors. In our work, we develop two types of deep neural network models mimicking kNN which are able to predict both the labels of closest samples and out-of-sample feature vectors at the same time. In experiments, our proposed models outperform the benchmark methods in both classification and oversampling tasks.

References

[Blackard and Dean, 1998] Jock A. Blackard and Denis J. Dean. Forest covertype. *UCI Machine Learning Repository*, 1998.

[Breiman, 2001] Leo Breiman. Random forests. *Machine Learning*, 2001.

[Carlsson *et al.*, 2008] Gunnar Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 2008.

[Chawla *et al.*, 2002] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 2002.

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing*, 2014.

[Chollet, 2015] François Chollet. Reuters newswire topics classification. https://keras.io/datasets", 2015.

[Cover and Hart, 1967] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 1967.

[Duarte and Hu, 2004] Marco F. Duarte and Yu Hen Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 2004.

[Erhan *et al.*, 2010] Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 2010.

[He *et al.*, 2008] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. *International Joint Conference on Neural Networks*, 2008.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, 2016.

[Hettich and Bay, 1999] S. Hettich and S. D. Bay. Network intrusion. *The UCI KDD Archive*, 1999.

[Hinton *et al.*, 2015] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *Annual Conference on Neural Information Processing Systems*, 2015.

[Karpathy, 2015] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

[Ke *et al.*, 2017] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and T. M. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Conference on Neural Information Processing Systems*, 2017.

[Krizhevsky *et al.*, 2009] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). http://www.cs.toronto.edu/~kriz/cifar.html, 2009.

[Lang, 1995] Ken Lang. Newsweeder: Learning to filter netnews. *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[LeCun and Cortes, 1999] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 1999.

[Maas *et al.*, 2011] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[Olah, 2014] Christopher Olah. Neural networks, manifolds, and topology. http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/", 2014.

[Sukhbaatar *et al.*, 2015] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. Weakly supervised memory networks. *Annual Conference on Neural Information Processing Systems*, 2015.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Annual Conference on Neural Information Processing Systems*, 2014.

[Wang *et al.*, 2017] Zhiguo Wang, Wael Hamza, and Linfeng Song. k-nearest neighbor augmented neural networks for text classification. *arXiv Repository*, 2017.

[Weston *et al.*, 2015] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *International Conference on Learning Representations*, 2015.

[Yeh and hui Lien, 2009] I-Cheng Yeh and Che hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 2009.

[Zoran *et al.*, 2017] Daniel Zoran, Balaji Lakshminarayanan, and Charles Blundell. Learning deep nearest neighbor representations using differentiable boundary trees. *arXiv Repository*, 2017.