Aalto University

Department of Neuroscience and Biomedical Engineering

Zero-shot classifier based on robust ellipsoid optimization

Author:

Kasper Rantamäki
kasper.rantamaki@aalto.fi

February 2023

Contents

1 Introduction

Machine learning has become a integral part of everyday life and especially in fields like text-generation huge leaps have been made of late. However, some fields lag significantly behind and these in general share the same reason for it - lack of access to large enough volume of data. One such field is neural language decoding as acquiring neurological data e.g. with MEG or fMRI is very expensive and the number of possible labels for which data is needed is excruciatingly large. Thus, it is only natural to wonder if predictions could be made to labels that the model hasn't seen in training and thus extend the capabilities of the model beyond the limits of the training data. The classic machine learning methods like k-nearest neighbor or multi-layer perceptrons are incapable of this, but a relatively new branch of machine learning called *zero-shot learning* attempts to answer this problem.

Using the definition from [1] we can define a zero-shot classifier or *semantic output code classifier* as:

Definition 1. *A semantic code classifier is a mapping $\mathcal{H} : F^n \to L$ defined as the composite of two functions - first mapping to a semantic space and the other from semantic space to label space - of form:*

$$\begin{aligned} \mathcal{H} &= (\mathcal{L} \circ \mathcal{S})(\cdot) \\ \mathcal{S} &: F^n \to S^p \\ \mathcal{L} &: S^p \to L \end{aligned}$$

where F^n is the feature space, S^p is the semantic space and L is the label space.

Essentially zero-shot learners attempt to provide some extra information in form of the semantic space to distinguish potentially unseen labels from those used in the training. One can think of this with the example that technically the model shouldn't have any trouble recognizing a zebra even if it has never seen one if it knows that zebra looks like a striped horse (and what a horse looks like) [wikipedia]. The problem then really is how to tell the model this required information. Common way to do this in neural language decoding is to use some vectors that are generated based on a large corpus and should carry relevant semantic information for a given word. Then one can make the assumption that points that are close to each other in the feature space are also close together in the semantic space. This is an assumption that will be used going forward.

2 Basic definitions

2.1 Probabilistic interpretation

Consider that we have two sets of datapoints X and Y, such that the points $\mathbf{x} \in X$ have the correct label and points in $\mathbf{y} \in Y$ some wrong one (can consist of datapoints with different labels as long as they are not the one with X). Each datapoint consists of features, which by they're nature follow some unknown, but distinct to the label, probability distribution. Thus, we need a robust classifier that takes the differing distributions into account. We now assume that the points follow distributions with finite variance and thus majority of the points should inhabit a finite area of the feature space. With these assumptions it becomes clear that we need not divide the whole feature space into segments, but allocate a smaller regions for each label. Clearly as each feature follows it's own distribution we can model the set X with a vector of random variables $\mathbf{X} = [X_1,\ X_2,\ ...,\ X_n]^T$, where n is the dimension of the feature space. The dependencies of the random variables in $\mathbf{X}$ can be studied with the covariance matrix:

Definition 2. *Covariance matrix $K \in \mathbb{R}^{n\times n}$ of a vector of random variables $\mathbf{X}$ is a symmetric matrix such that at:*

$$K_{ij} = K_{ji} = \mathrm{Cov}[X_i, X_j] \quad \forall\, i, j \in \{1,\ 2,\ ...,\ n\} \tag{1}$$

Identically this can be expressed as:

$$K = \mathrm{Cov}(\mathbf{X}, \mathbf{X}) = \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T] \tag{2}$$

Due to the properties of covariance we further know some properties of the covariance matrix K, like crucially that it is a symmetric and positive semi-definite:

Theorem 3. *Any given covariance matrix K is symmetric positive semi-definite.*

Proof. Symmetricity of the covariance matrix is a result of the commutative property of the covariance function. Given symmetric matrix $K \in \mathbb{R}^{n\times n}$ is positive semi-definite iff:

$$\mathbf{v}^T K \mathbf{v} \geq 0\ \forall\ \mathbf{v} \in \mathbb{R}^n \tag{3}$$

Using equation 2 and any arbitrary vector $\mathbf{v} \in \mathbb{R}^n$ we get:

$$\begin{aligned}\mathbf{v}^T K \mathbf{v} &= \mathbf{v}^T \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T]\mathbf{v} \\ &= \mathbb{E}[\mathbf{v}^T(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T\mathbf{v}] \\ &= \mathbb{E}[\|\mathbf{v}^T(\mathbf{X} - \mathbb{E}[\mathbf{X}])\|^2] \geq 0\end{aligned}$$

□

With some additional assumptions we can make sure the covariance matrix is not only positive semi-definite, but positive definite:

Theorem 4. *Covariance matrix $K \in \mathbb{R}^{n\times n}$ is symmetric positive definite (s.p.d) iff it is not singular (meaning it is invertible):*

Proof. Pending □

2.2 Modelling from samples and minimum volume ellipsoids

All of the above covered theory provides us a framework for defining the region containing all points in X that best models the probability distributions in $\mathbf{X}$. Indeed, if the covariance matrix $K \in \mathbb{R}^{n\times n}$ is s.p.d we can use it to define an ellipsoid:

Definition 5. *A closed ellipsoid in $\mathbb{R}^n$ centered at $\mathbf{c} \in \mathbb{R}^n$ with characteristic matrix $A \in \mathbb{R}^{n\times n}$ s.p.d is given by the set of points:*

$$\mathcal{E}_A(\mathbf{c}) = \{\mathbf{x} \in \mathbb{R}^n : (\mathbf{x} - \mathbf{c})^T A(\mathbf{x} - \mathbf{c}) \leq 1\} \tag{4}$$

Note that the characteristic matrix carries a lot of useful information as the eigenvectors of it are the principal axes of the ellipsoid and the eigenvalues are the reciprocals of the squares of the semi-axes [citation needed].

The covariance matrix should provide an ideal shape for the ellipsoid, but it might not enclose all points in X as it is ambiguous how many standard deviations away the points are. Thus, the covariance matrix should be scaled by some positive scalar c. Note that as per definition 5 the semi-axe lengths are tied to the eigenvalues and as per the eigenvalue equation $A\mathbf{v} = \lambda\mathbf{v}$ scalar multiplication of A clearly directly scales the eigenvalues. This means that given the covariance matrix K and optimal coefficient c we can formulate the minimum volume ellipsoid around the set of points X:

Definition 6. *Minimum volume ellipsoid (MVE) is the ellipsoid with smallest volume that still covers all wanted points. This is typically formulated as optimization problem:*

$$\begin{aligned} \text{min.} \quad & V(\mathcal{E}_{\mathbf{c}}(A)) \\ \text{s.t.} \quad & (\mathbf{x}-\mathbf{c})^T A(\mathbf{x}-\mathbf{c}) \leq 1 \quad \forall\, \mathbf{x} \in X \end{aligned} \tag{5}$$

where $V(\cdot)$ is the volume of a given input. The volume could be computed from either the Frobenius norm or the determinant of A.

In the case where the covariance matrix K is known and s.p.d the problem defined in 6 is simple as we can place $A = cK^{-1}$ and optimize for c. However, when computing the *sample covariance matrix* from given data the amount of data plays a crucial role as outlined in the following theorem:

Theorem 7. *A sample covariance matrix $K \in \mathbb{R}^{n\times n}$ is positive definite iff*

$$\operatorname{rank}\{\mathbf{x}_1-\mathbf{c},\ \mathbf{x}_2-\mathbf{c},\ \ldots,\ \mathbf{x}_{|X|}-\mathbf{c}\} = n \tag{6}$$

Proof. Extension of the proof in 4. □

This means that to find a s.p.d covariance matrix requires at least as many datapoints as is there is features in each datapoint. Understandably as the number of features grows very large acquiring the needed amount of data becomes infeasible in reality.

Alternatively, the definition of MVE can be used directly to find an approximation of an optimal ellipsoid even with smaller sample of data, but with the trade-off that the number of decision variables in the problem is significantly larger and some additional constraints are needed to maintain A as s.p.d. However, this approach is still used in some clustering based algorithms [citation needed].

3 Model definition and training

Even though MVE's can be used to form a classifier on their own, as they only consider points in the set X, that is the points with the correct label, the classifier wouldn't be all that robust. Robustness is crucial as we work with data that is stochastic in nature. Thus, instead of finding the ellipsoid with least volume, we would like to find the ellipsoid of which surface is the furthest distance from both the points in X and points in Y (set of points with wrong label), while enclosing as many points in X as possible. This can be alternatively formulated as finding the ellipsoid that maximizes the distance to the decision boundary for the closest points in both sets X and Y.

This is very similar to the definition used in support vector machine (SVM) algorithm [citation needed], with only the slight difference that we are attempting to find an ellipsoid rather than a hyperplane to separate the sets of points. And indeed we can modify the SVM algorithm to fit our purpose. Also do note that the following is just one way of accomplishing this task. An alternative (and a more commonly used one) would be to use what is called a kernel trick, which projects the datapoints into a higher dimensional space, where a hyperplane could be found to separate the sets of points while retaining that in the original space the separation is in form of an ellipsoid [citation needed].

3.1 Modified SVM algorithm

By basic principle we would like for the classifier to provide one signal for all datapoints in X and another for all datapoints in Y. Considering the ellipsoid $\mathcal{E}_A(\mathbf{c})$ defined in 5 if it is chosen so that it encompasses all points $\mathbf{x} \in X$, but not $\mathbf{y} \in Y$ would hold:

$$(\mathbf{x}-\mathbf{c})^T A(\mathbf{x}-\mathbf{c}) - 1 \leq 0 \quad \text{and} \quad (\mathbf{y}-\mathbf{c})^T A(\mathbf{y}-\mathbf{c}) - 1 \geq 0 \tag{7}$$

But to optimize for A requires some way of penalizing the wrong classifications and rewarding correct ones. Firstly we will want to introduce a buffer zone to add some robustness by rewriting equations in 7 in form:

$$(\mathbf{x}-\mathbf{c})^T A(\mathbf{x}-\mathbf{c}) - 1 \leq -1 \quad \text{and} \quad (\mathbf{y}-\mathbf{c})^T A(\mathbf{y}-\mathbf{c}) - 1 \geq 1 \tag{8}$$

Then we can relax this hardened constraint by introducing non-negative variables $u_1, ..., u_{|X|}$ and $v_1, ..., v_{|Y|}$ turning the equations from 8 into form

$$\begin{aligned} (\mathbf{x}_i-\mathbf{c})^T A(\mathbf{x}_i-\mathbf{c}) &\leq u_i, & \forall i \in \{1, ..., |X|\} \\ (\mathbf{y}_i-\mathbf{c})^T A(\mathbf{y}_i-\mathbf{c}) - 2 &\geq -v_i, & \forall i \in \{1, ..., |Y|\} \end{aligned} \tag{9}$$

The variables u_i and v_i measure how far from the correct classification the corresponding datapoints $\mathbf{x}_i$ and $\mathbf{y}_i$ fall. One can get an intuition for this by noting that when any given u_i or v_i is one then we have returned from the buffered definition in 8 to the non-buffered one in 7. And identically if the given u_i or v_i is greater than one the classification will be wrong. Thus our objective would be to minimize the sum of these measures.

The classifier definition is thus far identical to that of traditional SVM, but lacks the additional squared norm term associated with SVM. What this term does is pivot the hyperplane in hopes of finding an orientation that increases the distance to the closest points in both sets. Understandably, similar term doesn't really exist for ellipsoids, especially as they are surrounded on all sides by the points in set Y. Of course the size or shape of the ellipsoid could be controlled by e.g. the Frobenius norm of the characteristic matrix, but this doesn't take the points in Y into consideration in a meaningful way and is thus really no better than what MVE's could accomplish. Hence we will leave the additional term out of the objective function, meaning we can formulate a nonlinear optimization problem for finding the optimal ellipsoid as:

$$\begin{aligned}
\text{min.} \quad & \sum_{i=1}^{|X|} u_i + \sum_{i=1}^{|Y|} v_i \\
\text{s.t.:} \quad & u_i - (\mathbf{x}_i - \mathbf{c})^T A (\mathbf{x}_i - \mathbf{c}) \geq 0, && \forall i \in \{1, ..., |X|\} \\
& v_i + (\mathbf{y}_i - \mathbf{c})^T A (\mathbf{y}_i - \mathbf{c}) - 2 \geq 0, && \forall i \in \{1, ..., |Y|\} \\
& u_i \geq 0, && \forall i \in \{1, ..., |X|\} \\
& v_i \geq 0, && \forall i \in \{1, ..., |Y|\}
\end{aligned} \tag{10}$$

The used centerpoint $\mathbf{c} \in \mathbb{R}^n$ can be found as the the arithmetic mean of the datapoints $\mathbf{x} \in X$. Also note that additional constraints may need to be added to keep A symmetric positive definite.

To evaluate the performance and convergence of possible applied nonlinear optimization solvers we should know more about the nature of the problem at hand. And indeed we can show that the problem is convex:

Theorem 8. *The optimization problem defined in 10 is convex.*

Proof. Values u_i and v_i can be considered as affine functions and are thus both convex and concave. As a sum of convex functions with some multiplicative constants the objective function is convex. The constraints are either affine or sum of quadratic functions and affine functions and are similarly convex, meaning that the problem is a convex optimization problem as it consists of optimizing a convex function over a convex set [citation needed]. □

3.2 Model training

The above problem formulation doesn't yet make it clear how we can arrive into a zero-shot classifier, but it is important to remind oneself about the definitions of the sets X and Y as only thing we care is that the points in set X share the same label l_x of interest while Y contains all other points with labels $l_y \in L_0 \setminus \{l_x\}$. Here we also make a crucial distinction for zero-shot learning between the set of initial labels L_0 for which we have training data and the total set of labels L. Obviously must then hold that $L_0 \subset L$. Using these definitions we can divide the initial set of all datapoints F into sets X and Y for any arbitrary choice of $l \in L_0$ and thus optimize the ellipsoid according to 10 for all labels independently. This also means that this training is massively parallelizable.

4 Classification

The problem with allocating only closed regions of the feature space for the labels is that we will end up with $|L_0|+1$ possible classifications, while only having $|L_0|$ labels. Thus, the training doesn't find an unambiguous mapping in an off itself, but requires a secondary mapping function to create the unambiguity. Independent of whether we want to make zero-shot predictions or generic ones we would like to use as much of the information we have found in the training phase and we would like for the classifier to require minimal change when using the generic classification instead of the zero-shot one. In all such cases the classification would require finding some form of approximation for the label in question.

4.1 Zero-shot classification

There are many ways to find the approximation, but the most straight forward would be as the weighted average of the semantic vectors associated with the existing ellipsoids. The weight should be somehow inversely proportional to the distance from the surface of the ellipsoid to the point in question i.e. ellipsoids closer to the point should get a greater weight than those further from the point. Firstly we must define our distance metric. One intuitive metric would be the shortest Euclidean distance from each ellipsoid to the point. This could be computed using following theorem:

Theorem 9. *The shortest distance from an ellipsoid to a given point* $\mathbf{p} \in \mathbb{R}^n$ *outside of the ellipsoid can be computed by finding the closest point* $\mathbf{x}_c \in \mathcal{E}_A(\mathbf{c})$ *in the ellipsoid by noting that at this point the normal passes through the given point i.e the normal vector of the ellipsoid at the closest point* $\mathbf{n}_c$ *and the vector between the closest point and the point of interest* $\mathbf{u}_c = \mathbf{p} - \mathbf{x}_c$ *are parallel* $\mathbf{n}_c \parallel \mathbf{u}_c$.

Proof. Pending. □

However, with the distance metric defined in 9 the distance to the ellipsoid might be less clear if the point actually falls *within* the ellipsoid and even more so if it falls within multiple ellipsoids. Thus a better option might be to use the metric from the definition of the ellipsoid in 5 itself. This is actually the squared *Mahalanobis distance* from a distribution approximated by the inverse of the characteristic matrix A:

Definition 10. *Given a probability distribution* $\mathbf{X} \in \mathbb{R}^n$ *with mean* $\mathbf{c} \in \mathbb{R}^n$ *and covariance matrix* $K \in \mathbb{R}^{n\times n}$ *s.p.d. the Mahalanobis distance to some arbitrary point* $\mathbf{v} \in \mathbb{R}^n$ *from* $\mathbf{X}$ *is:*

$$d_M(\mathbf{v},\ \mathbf{X}) = \sqrt{(\mathbf{v}-\mathbf{c})^T K^{-1} (\mathbf{v}-\mathbf{c})} \tag{11}$$

If we use the notation from equation 7 we will end up with negative value for points within the ellipsoid and positive for those outside of it meaning that taking the inverse of the values doesn't provide the greatest value for the points in ellipsoids. Instead if we have a vector of distances $\mathbf{d} \in \mathbb{R}^{|L_0|}$ we can compute the wanted new values as:

$$\mathbf{w} = [\max(\mathbf{d}) - d_1 \quad \ldots \quad \max(\mathbf{d}) - d_{|L_0|}]^T \tag{12}$$

These values can then be passed through the *softmax* function to provide the usable weights that sum up to one.

Definition 11. *The softmax function is a generalization of logistic function and can be used to convert a vector of real values* $\mathbf{z} \in \mathbb{R}^m$ *into a probability distribution of* m *outcomes i.e.* $\sigma : \mathbb{R}^m \to (0,1)^m$*:*

$$\sigma(\mathbf{z})_i = \frac{\exp z_i}{\sum_{j=1}^{m} \exp z_j} \quad \forall i \in \{1, ..., m\}$$

Given that we are able to find the approximate semantic vector $\mathbf{s}_0$ for the given point using above described methods we should be able to find an approximate label. Again there is a multitude of ways to approach this secondary problem, but the simplest is to find the 1-nearest neighbour $\mathbf{s}_c$ from the set of semantic vectors S and return the label associated with it $l_c \in L$. With a small enough set of semantic vectors this could be done with brute force, but as the set grows more eloquent algorithms might be needed.

4.2 Generic classification

The zero-shot capabilities arise from the intermediary mapping to the semantic space by computing the weighted average of the semantic vectors associated with each ellipsoid. However, if access to semantic vectors proves difficult they could be exchanged with vectors representing the centerpoints of the ellipsoids, thus giving a robust approximation for the centerpoint of the ellipsoid the point should belong to. This can then be compared with existing centerpoints with 1-nearest neighbor search to reach an unambiguous classification.

5 Implementation

Current implementation is done mainly with Python for ease of integration with most data processing pipelines. Current working directory can be found on GitHub. Implementation uses some generic third party libraries Numpy, Scipy, Matplotlib etc. and is parallelized to multiple CPU core with Python's multiprocessing library [citations needed]. Additionally I have implemented the optimiser part in Julia using JuMP and the Ipopt solver [citations needed].

5.1 Performance optimization

The problem formulation from 10 is technically correct, but sub-optimal in practice. Main performance limitation comes from the large number of decision variables needed for the characteristic matrix A. The use of full matrix also dictates the need for additional constraints that further complicates the optimization. However, if we make the assumption that the features are independent of each other the only thing we would need to model are the variances of each feature, which are located on the diagonal of the characteristic matrix. This is an assumption that has been made in the current implementation as it greatly simplifies the problem.

Other time consuming operation is the matrix multiplication that needs to be done in the constraints as even with fast algorithms like Strassen's algorithm the time complexity is $\mathcal{O}(n^{\log_2(7)}) \approx \mathcal{O}(n^{2.81})$ [citation needed]. But when multiplying with a diagonal matrix most of the time is spent multiplying with zeros and instead we could do a Hadamard product (element-wise product) between a vector consisting of the diagonal elements of A and the difference vector of the datapoint and centerpoint.

With addition to the time complexity of the operations one should consider how the data is split. As the set X only contains points for a single label and Y for multiple, it is fair to assume that $|Y| \ggg |X|$. This will lead to an issue as it becomes much more important for the solver to minimize the significantly larger sum $\sum_{i=1}^{|Y|} v_i$. Thus we should have some multiple ω that increases the importance of sum $\sum_{i=1}^{|X|} u_i$. Trivial choice would be $\omega = |Y|/|X|$ giving the sums equal weight. However, after some point adding more points to the set Y doesn't have significant impact accuracy-wise, but with the increased number of decision variables v_i will have a significant impact to the computational complexity. Thus it might be wise to only use a random sample Y_0 of the set Y. The size of this random sample could then be chosen to be $\omega|X|$ with some user provided ω.

We get a modified problem formulation of form:

$$
\begin{aligned}
\text{min.} \quad & \omega \sum_{i=1}^{|X|} u_i + \sum_{i=1}^{|Y_0|} v_i & \\
\text{s.t.:} \quad & u_i - (\mathbf{x}_i - \mathbf{c})^T (\mathbf{a} \circ (\mathbf{x}_i - \mathbf{c})) \geq 0, & \forall i \in \{1, ..., |X|\} \\
& v_i + (\mathbf{y}_i - \mathbf{c})^T (\mathbf{a} \circ (\mathbf{y}_i - \mathbf{c})) - 2 \geq 0, & \forall i \in \{1, ..., |Y_0|\} \\
& u_i \geq 0, & \forall i \in \{1, ..., |X|\} \\
& v_i \geq 0, & \forall i \in \{1, ..., |Y_0|\}
\end{aligned}
\tag{13}
$$

where $\operatorname{diag}(A) = \mathbf{a}$.

5.2 Limitations

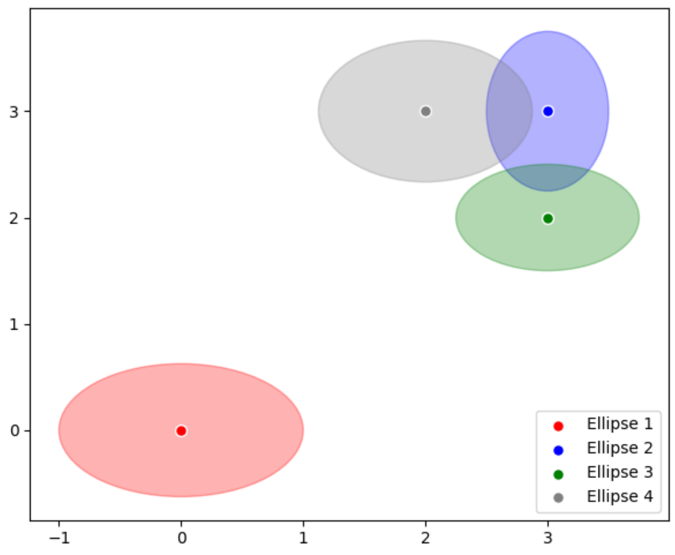

Figure 1: Example ellipses with a clear outlier

Even before testing the algorithm we can note some theoretical limitations it will have, especially considering the zero-shot capabilities of it. Largest issue comes when trying to zero-shot predict some outlier. Consider for example the ellipses found in figure 1. If we had as training data the labels "Ellipse 2-4" and had to predict a point that falls in the region defined by "Ellipse 1" the model would most certainly fail as the approximated semantic vector would be the average of the semantic vectors associated with labels "Ellipse 2-4" and thus should be far closer to them than the outlier (assuming a good choice of semantic space).

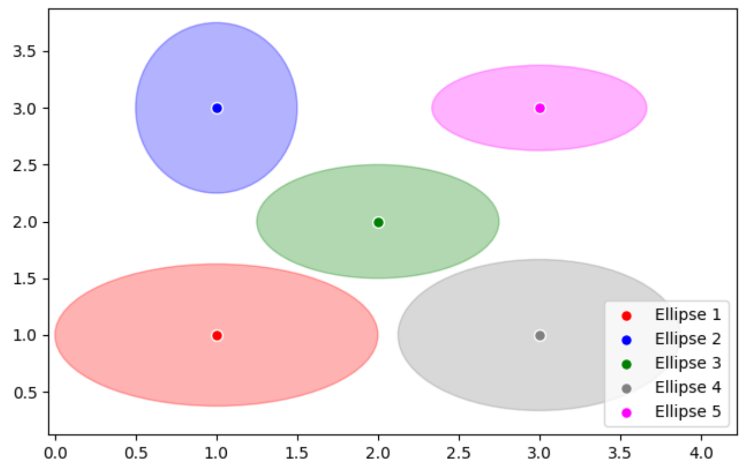

Figure 2: Example ellipses without a clear outlier

On the other extreme, where a good prediction would be almost guaranteed would be with a situation found in figure 2. Here if the training data consisted of labels ”Ellipse $\{1, 2, 4, 5\}$” and we had to predict a point that falls within the ”Ellipse 3” we should have a good chance as each of the training ellipses gets roughly an equal weight placing the approximate semantic vector pretty close to the one associated with ”Ellipse 3”. Of course if we tried to zero-shot predict some other label we would run to the same issues as discussed above.

In general then we would like to have as many labels as possible in the training data as then the probability that an unseen label is an outlier decreases and we can expect better zero-shot performance.

6 Results

We will test the outlined algorithm in two cases. Firstly we will test it with randomized points and mainly study how the choice of semantic vectors and number labels left out of the training data affect the prediction accuracy. We compare the algorithm with k-nearest neighbour (KNN) algorithm [citation needed] and generic version of the outlined algorithm.

Secondly we will test the generic algorithm with a real (although simple) dataset and similarly compare the performance to KNN algorithm. KNN is chosen as comparison since it similarly depends on points for a given label to be nearby in feature space and can work with relatively small number of training points unlike some more complicated classifiers. In both cases we choose parameters to be $k = 15$ for the KNN algorithm and $\omega = 10$ for our algorithm.

6.1 Randomized points

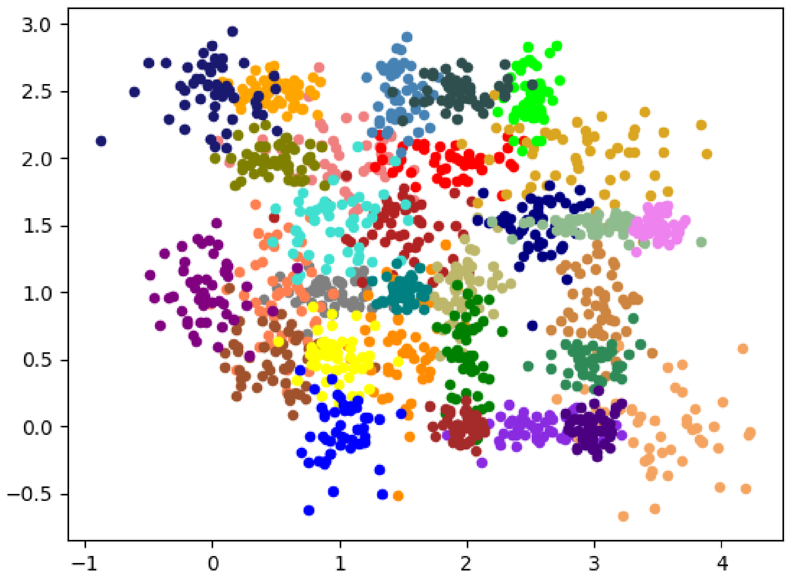

Figure 3: 50 randomly generated points for each label outlined in table 1

Our randomized data consists of random datapoints each with features acquired from a normal distribution with the mean and variance of the associated label. The sets of means and variances can be found in table 1. There are 30 of such sets meaning we have 30 labels to use. For visualization purposes the sets consists of only two means and variances. We generated 50 datapoints for each label for training and similarly 50 datapoints per label for testing. One example of the training points can be seen in figure 3. It is clear that the classification problem isn't exactly trivial.

Table 1: Means and standard deviations of the labels used

μ_x	μ_y	σ_x^2	σ_y^2
1.00	1.00	0.20	0.20
1.00	2.00	0.30	0.30
1.50	1.50	0.30	0.30
2.00	2.00	0.30	0.30
0.50	1.00	0.20	0.20
0.50	0.50	0.20	0.20
3.00	1.00	0.20	0.20
1.50	0.50	0.20	0.20
3.00	2.00	0.40	0.40
2.00	1.00	0.15	0.15
0.50	2.00	0.20	0.20
1.00	0.50	0.15	0.15
2.00	0.50	0.10	0.10
2.50	2.50	0.10	0.10
1.00	1.50	0.30	0.30
1.50	1.00	0.10	0.10
1.50	2.50	0.15	0.15
2.50	1.50	0.20	0.20
1.00	0.00	0.15	0.15
2.50	0.00	0.30	0.30
0.00	1.00	0.20	0.20
2.00	0.00	0.10	0.10
3.50	0.00	0.30	0.30
0.50	2.50	0.20	0.20
3.00	1.50	0.30	0.30
3.00	0.50	0.20	0.20
2.00	2.50	0.20	0.20
0.00	2.50	0.30	0.30
3.00	0.00	0.10	0.10
3.50	1.50	0.10	0.10

With randomized data an obvious question becomes what to use as the semantic vectors. In this case we have two different mappings for the means that map to a higher dimension. Firstly we have a relatively complicated nonlinear mapping of form:

$$\mathrm{sem}_{\mathrm{nl}}(\mu_x,\ \mu_y) = [\mu_x^2 - \mu_x \quad \mu_x^{1/2} + \mu_y^{3/2} \quad 2\mu_y - \mu_x^2 \quad \mu_y^3]^T \tag{14}$$

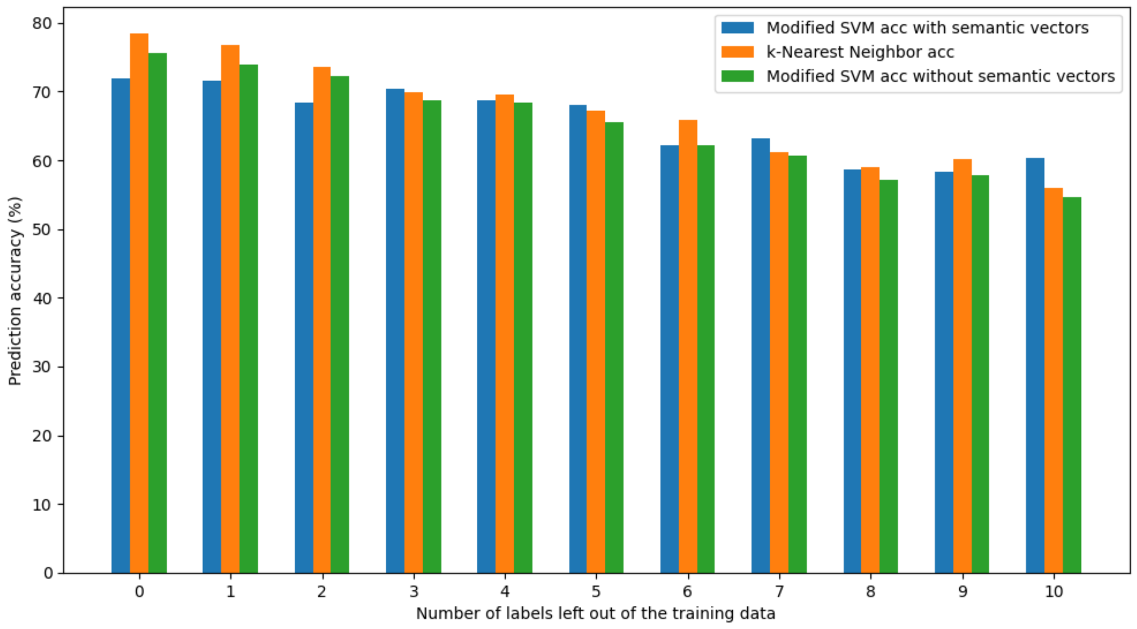

Figure 4: Testing results with semantic vectors defined by mapping in 14

The results of training and testing with semantic vectors defined by mapping in 14 are shown in figure 4. It is clear that the performance isn't all that great. In general it doesn't seem that the semantic vectors provide much extra information to the classifier as the generic version is able to hold it's own even with multiple left out labels and neither equals the predictive accuracy of the KNN algorithm. This result is not necessarily that surprising as the mapping for semantic vectors only has two unambiguous values for any given mean and thus might not represent the feature space that accurately.

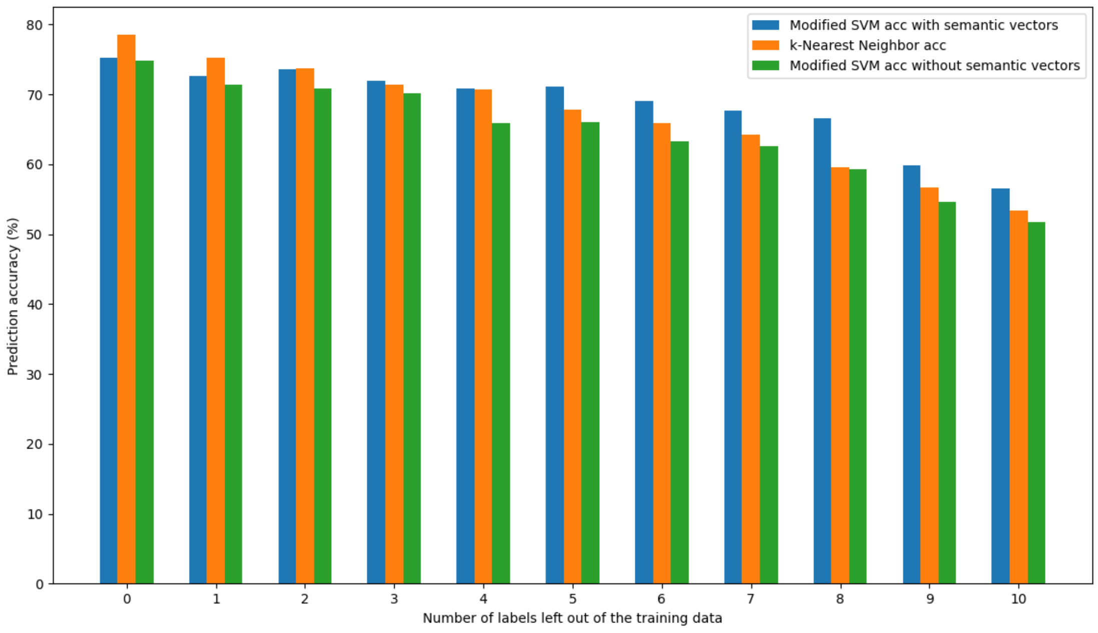

Figure 5: Testing results with semantic vectors defined by mapping in 15

Second mapping we define is far simpler and linear in nature, which should better maintain relative locations in the semantic space:

$$\mathrm{sem}_l(\mu_x,\ \mu_y) = [3\mu_x + 3.2 \quad 0.761\mu_x \quad 0.2335\mu_y \quad 7\mu_y + 1.245]^T \tag{15}$$

The results of training and testing with semantic vectors defined by mapping in 15 are shown in figure 5. This mapping performs understandably much better. Indeed, it out-competes the generic classifier in every case and KNN algorithm in most cases. The most impressive part however is how with the semantic information the algorithm is able to maintain almost constant prediction accuracy as the number of left out labels grows. In contrast for both KNN and the generic classifier the prediction accuracy decreases in a very linear fashion.

6.2 MNIST dataset

In the second test we use the MNIST (Modified National Institute of Standards and Technology) dataset [citation needed]. This dataset consists of 60 000 training images and 10 000 testing images of handwritten digits, which are converted into black and white and normalized to fit into 28 × 28 pixels. This is a good choice for testing as it is very clear how each handwritten digit both follows some unknown probability distribution and should inhabit only a finite region in space. We will use a random sample of 25 points per label for training and 75 points per label for testing. The sampling is done a total of five times and the associated results are found in table 2.

Table 2: Prediction accuracies for our algorithm and KNN for 5 different samples with 25 training points and 75 testing points per label

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
Mod SVM	0.572	0.536	0.511	0.477	0.469
KNN	0.715	0.705	0.724	0.712	0.708

The result is by no means spectacular, but the prediction accuracy is still significantly above chance and for such a small number of training points relatively decent. However, the problem really is that increasing the number of training points doesn't have that much of an impact on the result for our algorithm while the performance for KNN does improve as is seen in the table 3.

Table 3: Prediction accuracies for our algorithm and KNN for 5 different samples with 50 training and testing points per label

	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
Mod SVM	0.548	0.504	0.566	0.562	0.524
KNN	0.802	0.786	0.802	0.772	0.808

The main reason for the poor performance is due to our limiting implementation. Currently the implementation hinges on the assumption that all features are independent, but in the case of MNIST dataset this isn't the case. Consider for example the number one. Generally it is characterized as a straight line (with possibly a hook at the top) at a slight angle. Thus, technically all pixel values for a handwritten number one could be inferred from knowing just two (relevant) pixel values. With MNIST as the digits are normalized and hence centered we can assume that the handwritten number one will pass through the centermost pixel and if we happen to know the top most pixel for it we already have the needed two points and thus all other pixels should be highly correlated with them. This means that the assumption of independence restricts our predictive ability drastically and if the implementation were to be changed to one with filled characteristic matrix we should see far greater prediction accuracies. Similar can be assumed to hold for all other digits.

7 Discussion

Pending

References

[1] Palatucci, M., Pomerleau, D., Hinton, G. E., Mitchell, T. M. (2009). Zero-shot learning with semantic output codes. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, A. Culotta (Eds.), Advances in Neural Information Processing Systems (Vol. 22, pp. 1–9). Curran Associates, Inc.