Zero-shot decoder based on modified SVM

Kasper Rantamäki
February 2023

1 Introduction (under construction)

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 *zero-shot learning* attempts to answer this problem.

Using the definition from [palatucci] 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 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 Math (placeholder)

2.1 Dataset and basic definitions

Consider that we have two sets of datapoints X and Y, such that the points in X have the correct label and points 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 are of the feature space. With these assumptions it becomes clear that we need not divide the whole feature space into segments, but allocate smaller regions for each label. The shape of the allocated space can be debated endlessly, but what is important is that the shape is closed. Simplest choice would be a ball:

Definition 2. *A closed ball in $\mathbb{R}^n$ centered at $\mathbf{c} \in \mathbb{R}^n$ with radius r is given by the set of points:*

$$\mathcal{B}_r(\mathbf{c}) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{c}\|_2 \leq r\}$$

Note that we use the definition given with L_2-norm as it is convenient going forward, but any choice L_p would be valid [citation needed].

However, a ball can end up having a lot of wasted space if the size of the variances varies greatly between the features. Thus a good choice for the shape could be the ellipsoid:

Definition 3. *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:*

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

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].

It is also good to note that as ellipsoid is the result of an invertible linear transformation to a ball (defined by the characteristic matrix A) we can also define a n-dimensional ball as:

Remark 4. *Using definition 2 a closed ball in $\mathbb{R}^n$ centered at $\mathbf{c} \in \mathbb{R}^n$ with radius r is given by the set of points:*

$$\mathcal{B}_r(\mathbf{c}) = \{\mathbf{x} \in \mathbb{R}^n : r^{-2}\|\mathbf{x} - \mathbf{c}\|_2^2 \leq 1\}$$

This also clearly derivable from definition 1.

This notion will prove useful as it is quadratic.

2.2 Problem formulation

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 2 if it is chosen so that it encompasses all points $\mathbf{x} \in X$, but not $\mathbf{y} \in Y$ would hold:

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

But to optimize for A requires some way of penalizing the wrong classifications and rewarding correct ones. To do this we use similar approach as with optimizing support-vector machines

(SVM) [citation needed], but instead of fitting a hyperplane between the sets we fit the ellipsoid. Firstly we will want to introduce a buffer zone to add some robustness by rewriting equations in 1 in form:

$$(\mathbf{x}-\mathbf{c})^T A(\mathbf{x}-\mathbf{c}) - 1 \leq -1 \quad \text{and} \quad (\mathbf{y}-\mathbf{c})^T A(\mathbf{y}-\mathbf{c}) - 1 \geq 1 \tag{2}$$

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 2 into form

$$\begin{aligned} (\mathbf{x}_i-\mathbf{c})^T A(\mathbf{x}_i-\mathbf{c}) &\leq u_i, && \forall i \in \{1, ..., |X|\} \\ (\mathbf{y}_i-\mathbf{c})^T A(\mathbf{y}_i-\mathbf{c}) - 2 &\geq -v_i, && \forall i \in \{1, ..., |Y|\} \end{aligned} \tag{3}$$

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 2 to the non-buffered one in 1. 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. For robustness sake we should also maintain smallest possible semi-axe lengths i.e. attempt to find the minimum volume ellipsoid for the problem. The constraints then come from equations in 3 and the fact that by definition variables u_i and v_i are non-negative. We get a problem of form:

$$\begin{aligned} \text{min.} \quad & \sum_{i=1}^{|X|} u_i + \sum_{i=1}^{|Y|} v_i + \gamma \|\mathbf{w}\|_2^2 \\ \text{s.t.:} \quad & u_i - (\mathbf{x}_i-\mathbf{c})^T A(\mathbf{x}_i-\mathbf{c}) \geq 0, && \forall i \in \{1, ..., |X|\} \\ & v_i + (\mathbf{y}_i-\mathbf{c})^T A(\mathbf{y}_i-\mathbf{c}) - 2 \geq 0, && \forall i \in \{1, ..., |Y|\} \\ & u_i \geq 0, && \forall i \in \{1, ..., |X|\} \\ & v_i \geq 0, && \forall i \in \{1, ..., |Y|\} \end{aligned} \tag{4}$$

where γ is a provided factor which controls the trade-off between classification accuracy and robustness and $\mathbf{w} \in \mathbb{R}^n$ is the vector holding the lengths of the semi-axes i.e. $\mathbf{w} = [\lambda_1^{-\frac{1}{2}} \quad \lambda_2^{-\frac{1}{2}} \quad ... \quad \lambda_n^{-\frac{1}{2}}]^T$ (λ_i being the i:th eigenvalue of A). Note that additional constraints need to be added to keep A symmetric positive definite. This problem can easily be converted into one optimizing for balls if such is wanted:

$$\begin{aligned} \text{min.} \quad & \sum_{i=1}^{|X|} u_i + \sum_{i=1}^{|Y|} v_i + \gamma r^2 \\ \text{s.t.:} \quad & u_i - r^{-2}\|\mathbf{x}_i-\mathbf{c}\|_2^2 \geq 0, && \forall i \in \{1, ..., |X|\} \\ & v_i + r^{-2}\|\mathbf{y}_i-\mathbf{c}\|_2^2 - 2 \geq 0, && \forall i \in \{1, ..., |Y|\} \\ & u_i \geq 0, && \forall i \in \{1, ..., |X|\} \\ & v_i \geq 0, && \forall i \in \{1, ..., |Y|\} \end{aligned} \tag{5}$$

Note that solving for the centerpoint of the ellipse (or ball) $\mathbf{c} \in \mathbb{R}^n$ is trivial as it is just the

arithmetic mean of the datapoints $\mathbf{x} \in X$

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 5. *The optimization problem defined in 4 (also identically in 7) is convex.*

Proof. Values u_i and v_i can be considered as affine functions and are thus both convex and concave and the squared norm of $\mathbf{w}$ (or r^2) is a quadratic term and thus convex. As a sum of convex functions with some multiplicative constants the objective function is convex. The constraints are either affine or sum of quadratic function 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]. □

2.3 Zero-shot 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 4 for all labels independently. This also means that this training is massively parallelizable.

2.4 Zero-shot prediction

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. However, this is not a problem for us since instead of directly mapping to the label space all we need to use the ellipsoids for is to find an approximation of the semantic vector for the given datapoint and pass it to the second mapping to find the actual label.

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 get greater weight than those further from the point. Firstly we must define our distance metric. If we wanted to be technically correct we should consider the shortest distance from each ellipsoid to the point. This could be computed using following theorem:

Theorem 6. *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 6 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 3. This has the benefits of being easy to compute, of providing a metric for points inside of ellipsoids as well as outside and as technically being the squared euclidean distance in the coordinate system stretched by A should be relatively accurate.

If we use the notation from equation 1 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:

$$\mathbf{v} = [\max(\mathbf{d}) - d_1 \quad ... \quad \max(\mathbf{d}) - d_{|L_0|}]^T \tag{6}$$

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

Definition 7. *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.

3 Implementation (under construction)

Current implementation is done with Python for an ease integration with most data processing pipelines like the ones in neural language decoding that use Python specific libraries e.g. MNE-Python. Current working directory can be found in my GitHub. Implementation uses some generic third party libraries Numpy, Scipy, Matplotlib etc. and is parallelized to multiple CPU core with Python's multiprocessing library.

3.1 Performance optimization

The problem formulation from 4 is technically correct, but sub-optimal in practice. Main performance limitation comes from the need to compute the eigenvalues of the characteristic matrix A for use in the objective function. Eigenvalues can be found with e.g. Schur decomposition [citation

needed], but this is a $\mathcal{O}(n^3)$ time complexity operation (n being the number of rows or columns in A) that would need to be done with each iteration of the solver. However, depending on the interpretation of the problem this might not be necessary. One needs to note that the values outside of the diagonal of A can realistically only rotate, twist or mirror the ellipsoid with respect to the coordinate system. Each axis of the coordinate system represents one feature in the feature space and thus the semi-axes of our ellipsoids should approximate the probability distributions for them individually. This is obviously not possible if the ellipsoid is rotated leading to the linear combination of multiple semi-axes to represent a singular probability distribution. As this is unwanted it would be enough for A to be a diagonal matrix as then the only effect it has is with stretching the semi-axis. Diagonal matrices as a subset of triangular matrices holds that the values on the diagonal are the eigenvalues meaning that accessing them is a constant time operation. Additionally, this reduces the number of needed decision variables drastically and thus makes the optimization problem significantly easier for the solver.

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 data point and center point.

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 leads 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.

Final modification to improve the user experience concerns the multiple γ. The weight given to the semi-axes lengths depends not only on γ, but the total number of data points as the sums grow asymptotically with the sizes $|X|$ and $|Y_0|$. Thus to have a equivalent weight independent of the number of data points the γ should be multiplied with $|X| + |Y_0|$.

We get a modified problem formulation of form:

$$
\begin{aligned}
\text{min.} \quad & \omega \sum_{i=1}^{|X|} u_i + \sum_{i=1}^{|Y_0|} v_i + (|X| + |Y_0|)\gamma \|\mathbf{w}\|_2^2 & \\
\text{s.t.:} \quad & u_i - (\mathbf{x}_i - \mathbf{c})^T (\mathbf{a} \circ (\mathbf{x}_i - \mathbf{c})) \geq 0, & \forall i \in \{1, ..., |X|\} \\
& v_i + (\mathbf{y}_i - \mathbf{c})^T (\mathbf{a} \circ (\mathbf{y}_i - \mathbf{c})) - 2 \geq 0, & \forall i \in \{1, ..., |Y|\} \\
& u_i \geq 0, & \forall i \in \{1, ..., |X|\} \\
& v_i \geq 0, & \forall i \in \{1, ..., |Y|\}
\end{aligned}
\tag{7}
$$

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

3.2 Limitations

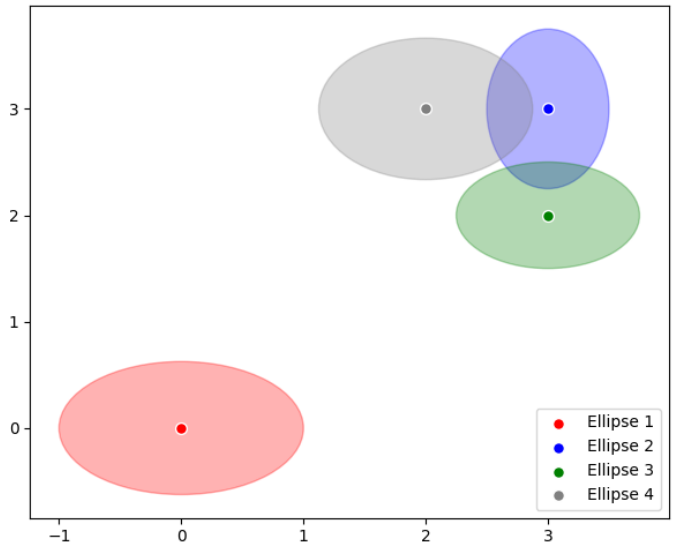

Figure 1: Example ellipses with a clear outlier

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).

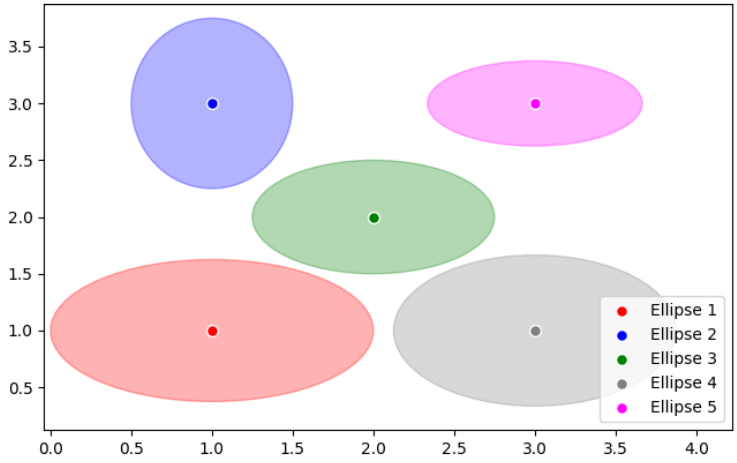

Figure 2: Example ellipses without a clear outlier

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.

4 Results

We will test the performance of the outlined algorithm in two cases: with randomized data that follows clear probability distributions and can thus be considered an ideal case and then with MEG data. In both cases the performance is compared with k-nearest neighbors algorithm [maybe some other as well?] as it works in a comparable way to our algorithm.

4.1 Randomized data

Our randomized data consists of random datapoints each with feature 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.

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 there is no good choice so for each label the associated semantic vector is computed as $[\mu_x^2 - \mu_x \quad \mu_x^{1/2} + \mu_y^{3/2} \quad 2\mu_y - \mu_x^2 \quad \mu_y^3]^T$. This choice is clearly maps the means in a non-linear way and maps them to a different dimension, which should be usable for testing purposes while not being trivial.

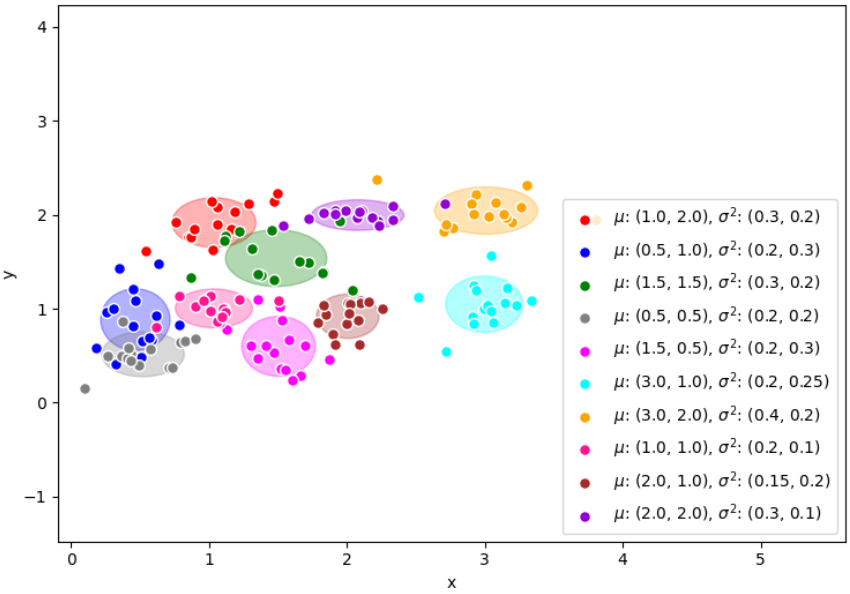

Figure 3: The ellipses found for the 10 first labels in table 1

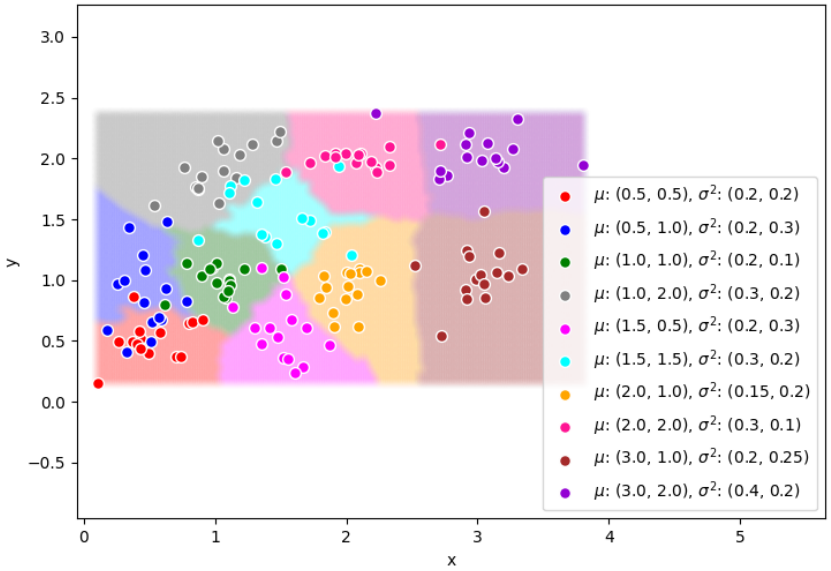

Figure 4: The k-nearest neighbors prediction regions found for the 10 first labels in table 1

We will generate 20 datapoints for each label and use 15 of these for training purposes and 5 for testing. With our algorithm we choose as parameters $\omega = 5$ and $\gamma = 1$ and with k-nearest neighbors we choose $k = 15$. The training results for the first 10 labels with both algorithms can be seen in figures 3-4. The results would vary somewhat if all 30 labels where included. Clearly we can see

that our classifier is more conservative with allocating regions of the feature space, which is to be expected. We could of course alter this by changing the γ value, but this seems reasonable.

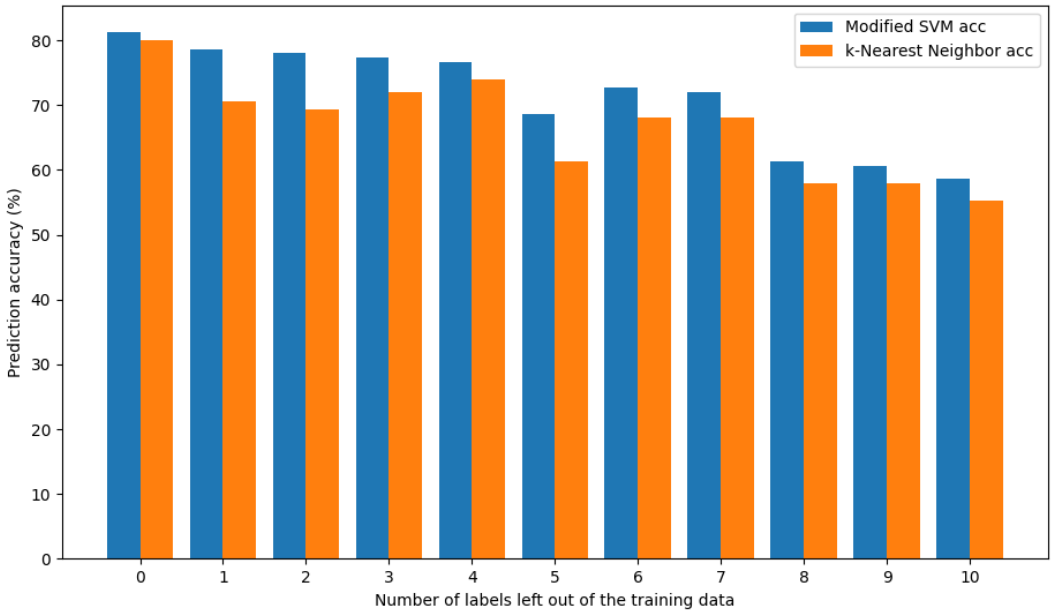

Figure 5: Leave-n-out accuracy of all labels in table 1 with different values of n

In figure 5 we can see the results of running our algorithm against k-nearest neighbors algorithm with different number of labels being left out of the training set. It appears that in general our algorithm has greater prediction accuracy, which isn't necessary that impressive in this ideal case. Our algorithm is also generally more consistent with the accuracy especially with a small number of left out labels, which is to be expected thanks to the zero-shot capabilities. However, as the number of left out labels increases the performance of our algorithm also decreases to not being much better than the k-nearest neighbors, meaning that most zero-shot predictions must have failed. This makes sense as then we have much less semantic vectors to average together worsening the approximation and increasing the chance of outliers.

4.2 MEG data

To be added