Zero-shot decoder based on modified SVM

Kasper Rantamäki

January 2023

1 Introduction (under construction)

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.

[Additional stuff]

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 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. Thus our objective would be to minimize the sum of these measures while also maintaining smallest possible semi-axe lengths. The constraints 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 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 5) is convex.*

Proof. Values u_i and v_i can be considered as affine functions and are thus both convex and convex 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 $l_y \in L_0/\{l_x\}$. 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 $\mathbf{p} \in \mathbb{R}^n$ $\underline{\text{outside of the ellipsoid}}$ 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 should be relatively accurate due to the linear transformation caused by the characteristic matrix [make clearer].

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.

2.5 Pseudocode

To be added

3 Implementation (under construction)

[Text pending, but working directory can be found on my GitHub]

Thus far I have only been able to make the classifier utilizing balls work and some initial results can be seen in figures 1-3

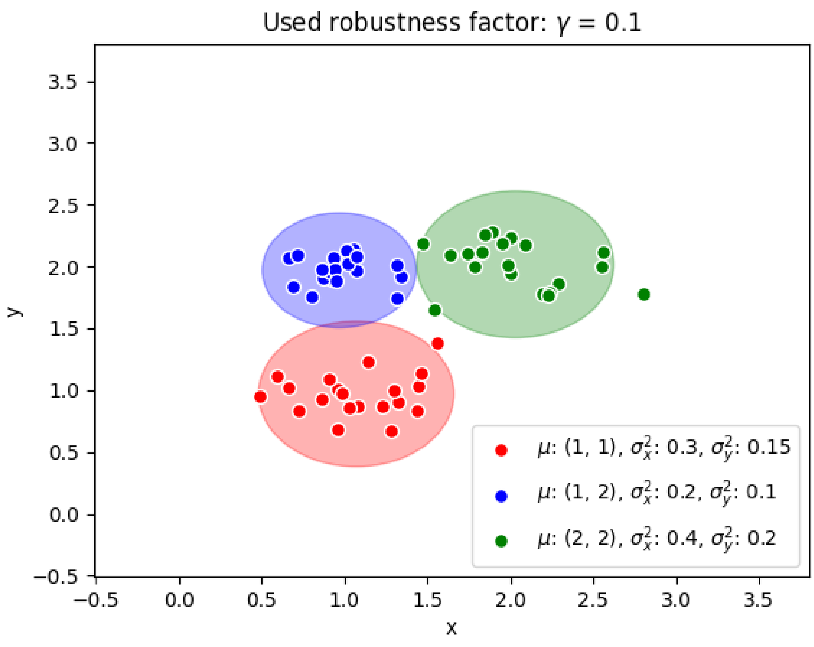

Figure 1: Optimization results for 3 labels each with 20 randomized datapoints with chosen γ value of 0.1

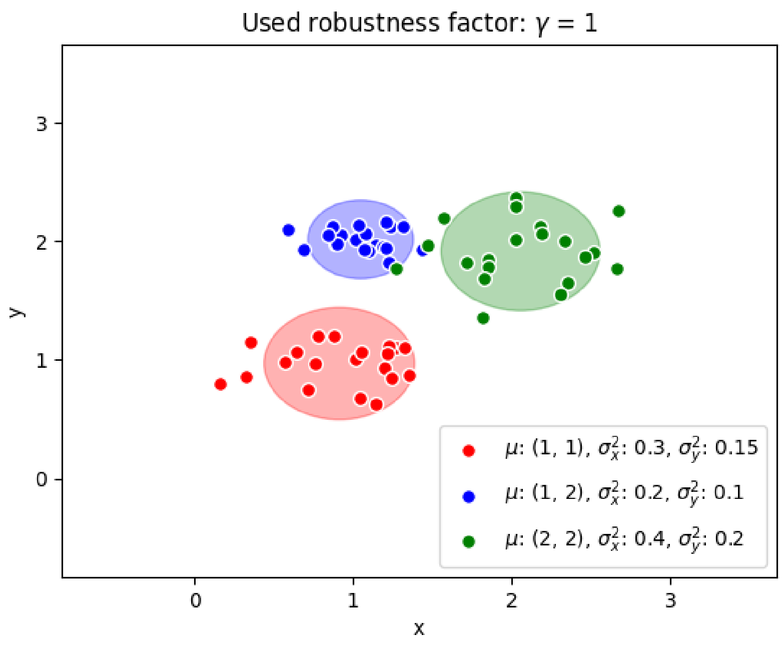

Figure 2: Optimization results for 3 labels each with 20 randomized datapoints with chosen γ value of 1

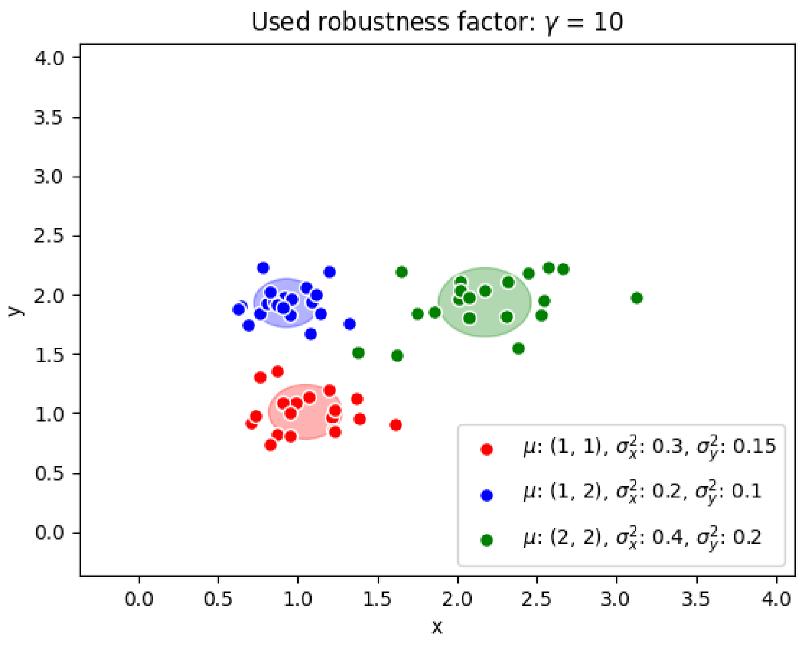

Figure 3: Optimization results for 3 labels each with 20 randomized datapoints with chosen γ value of 10