Handout: Linear Classifiers

Linear classifiers are a particular class of methods that are used in ML. LC have become very popular in LT in the last 15 years. DTs and knn thare are examples of non-linear classifiers.

Some examples of linear classifiers are maximum entropy models, support vector machines, perceptrons, etc.

Let's start with simple classification. The input can be a document or a sentence or some kind of linguistic entity. And the output space is a potential output, for ex a parse tree, a document class, a sequence of POSs…

Feature Representations

We assume that we can represent input and output as a feature vector. This is the first difference. So far we have assumed the input is represented by features, and we use some kind of algorithm to determine the class. In the case of DTs, we start at the root node and through tests we construct the tree, and then in the end we determine the class. We do not explicitly represent the class in the features. But in LC we have to define feature vectors over pairs of inputs and outputs.

Features and Classes -- Just to point out the main differences:

1) All features must be numerical. If we have features that are naturally numerical, like the word length, then no problem, we just use the numbers. If we have binary features thar represent true or false, we an also represent them as numeric features, using 0 and 1
2) if we have categorical features (multinomial), that are more than 2 (so cannot be handled as binary, we must devise a mapping to numerical features in order to used them in linear classifiers. What is mostly use in NLP is binarization. Instead of having a single feature x or y that can take on some number of discrete values… Think this is part of speech of some word of interest which can have 20 values for 20 POSs… instead of having that we now introduce 20 different features and let each of these features be a binary feature (0,1), and then you define them in such a way that a particular feature, jth feature, take 1 if and only if the original polinomial feature takes off the j value. Instead of having a single feature that takes off category and classes, we make sure that only one takes 1 and all the other are 0s.
3) (With categorical features we had many values for a single features, so they were more expressive.) it seems that you lose information doing this because in the actual model the features look like independent from each other, and the actual model is not going to know that they have nothing to do with each other In the training instances only one of them is going to be true or 1 at any given time. This is the basic approach that people use.
4) the other thing to know compared with the model we saw so far is that we need distinct features for distinct output classes. Assume now we have done the mapping and got rid of categorical features, instead of having a single feature that tells you that a pos is a verb or noun, we now have to take into account that we have n-outupt classes. And we need to define distinct feature for each of them. So we now have a pos that is verb and the output is person name or organization, we have to make sure that in this high dimensional space each of these features that we have for distinct value classes takes on the value of the original input feature if and only if the class that we give as input corresponds to the index of this feature.

Block Feature Vectors

This kind of representation is called the block vector, or block feature vector. And this is the standard representation when we need to do a multiclass classification.

You get one equal size block of the entire feature block for each label. In this case we have only 2 labels (person and object) so we have only 2 blocks and they are length 4 because we have 4 input features. But we can have any finite number of classes and any finite number of features. We basically duplicate any input feature in any block. A non zero value are allowed only in one block.

Depending on the label, we activate/fire the features in one block, and all the other ones are zero.

Contrast betw the block vector representation and the simple input vector representation. This property that we have non zero values only in one block is typical of simple multiclass classification where we have discrete atomic classes: every given instance has one class and not the others. But this does not always hold, as we will see next time…

This is the basic setup concerning feature representation: Same idea as previous classifiers, but different representation, because all the features must be numeric, and we need distinct features for different output classes.

The actual classification works as follows:

A linear classifier is so called because the score (sometimes interpreted as probability, but generally it is just a numerical score) that we assign is based on a linear combination of features and their weights.

if we have a feature vector of length m, then we also have a weight vector, where the weights reflects the significance of a particular feature. If a feature is indicative of a particular class, it should have a positive high value and correspondly low negative value.

how we are going to learn these weights? Assume for now that we know these weights, then the actual classification is very simple: if we a have a signed weight (that can be positive or negative) to each feature, then for the multiclass classification and classes, all we have to do is to compute for each possible output class the product of the feature vector and the weight vector. this is the so called dot product or inner product of the vector

The idea is that we compute the product for all these components and we sum them up. we first compute the products and then we sum the values. So we are going to sum a sequence of numbers, some of them are positive, some of them are negative. Some of them are 0, either because the weight is 0, or (more commonly) because the feature is 0. In the end we get a number, and this is the score assigned to a particular class y. And then in multiclass classification we get the class that gets the highest score. There is always going to be such a class. In some extreme cases, there might be no class that has such a score.

the binary classification is only a special case of multiclass classification, where one class with a score above 0 is the positive class

this is what a linear classification is: we have a feature vector classification where all features are numerical, we have a weight vector that reflect the significance of features and we perform the classification by performing the calculations and picking up the highest score.

If our weights are reasonable (ie high weights for indicative features), we could expect to get reasonable good results.

Linear Classifiers – Bias Terms: When linear classifiers are presented in the literature, they are presented in a slightly different form. They say that we should find a class that maximise the score, and the score is the inner product of the weight vector and the feature vector, then there is an additional term, called the bias term it shifts the dividing lines betw the classes with respect to the origin (we will see this later). The slope of line remains the same, but we can move it closer or away of the origin.

Actually we can get rid of this bias term by encoding this as another feature. It will be a feature only dependent on the class and not on the input, so if we extend our representation of George

Washington example, and we add a particular feature here in each block, the last feature is 1 only when the class is active, you see the red 1 here, then we can represent the value of the bias term of a particular class by the weight of that feature. This is a technicality, but it is in the case you wonder about the bias term

Binary Linear Classifier: What does a linear classifier do? In the case of a binary linear classifier, in the general case it defines the so called hyperplane . In the simplest case, the hyperplane is just a straight line. We can move it but it will always be straight. That's why it is called linear.

If we have a binary classification problem and we train a linear classifier, what we learn is a weight vector that defines a straight line in this space: one class is one side, the other class is on the other side.

Multiclass Linear Classifier: when we are doing multiclass classification is a little more complex but the regions of the space are defined by straight lines that assign different weights to features for different classes.

Separability: One notion that comes up is the notion of separability. A training set (a set of points) is said to be separable if there is a weight vector achieves perfect classification.

If we look at the a two dimensional problem, the set is separable, we can fit a line that perfectly separates positive and negative points.

If the set is non separable, no matter what weight vector we have, we can't find a way to separate them

How to find w: How to find weights of the vector w that gives us good accuracy on the training set and above all good generalization?

As input to solve this problem we are given 2 things: a training set T represented by pairs of input and output, and we have to decide on the feature representation f (this must be decided before).

we want to find is a weight vector that maximize or minimize (it depends on how you formulate this) some important functions of the training set. And this is where different kinds of linear classifier make different assumptions. This is where they have different inductive biases.

One idea that we find in several of these is to try to minimize errors on the training set, that is trying to find a weight vector that classify as many points as possible correctly. This is the case of perceptron, and with some variations in svm and boosting.

another idea is to maximize the likelihood or probability (see maximum likelihood estimation) and this is the case with logistic regression (discriminative) and naive bayes (generative) .

36:30 for most of the time today, we are going to assume that the training set is separable. In real life this is often not the case. There is a lot of literature on this. And even a training set is not perfectly separable, it is possible to find a good classifier that performs well on unseen data.

the end for now---