Perceptron

Perceptron is a simple machine learning #algorithm based on a linear model.

Algorithm

```
repeat until convergence:
        for each training example (f_1, f_2, ..., f_n, label):  # label = -1/1
                check if correct based on the current model
                if not correct, update all the weights:
                                for each w_i:
                                        w_i = w_i + f_i * label
                                b = b + label
```

The check on correctness could be performed assigning our prediction to a variable $\textit{prediction} = b + \sum_{i=1}^{n} w_i f_i$ and then checking this variable against the `label`, as in

```
if prediction ≠ label:
```

Which line does the perceptron find?

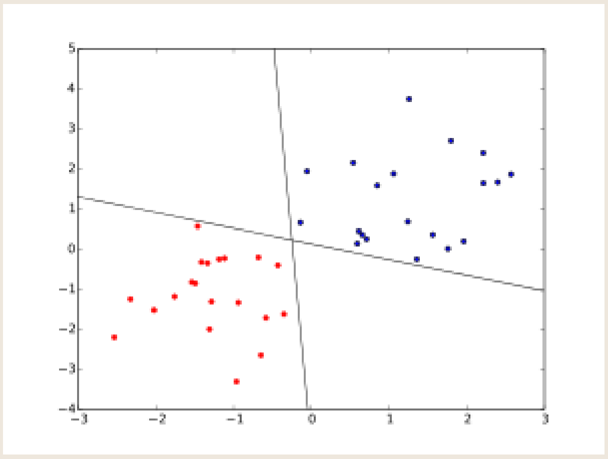
It isn't guaranteed that the perceptron will find the optimal line to satisfy the learning task, only that it will find *some* line that separates the data.

Note

If the data we're learning from isn't linearly separable the algorithm clearly won't #converge , in which case an approach could be to impose a limit on the number of iterations. If we encounter this problem, though, we should really be asking ourselves if using a linear model is a good idea.

Note

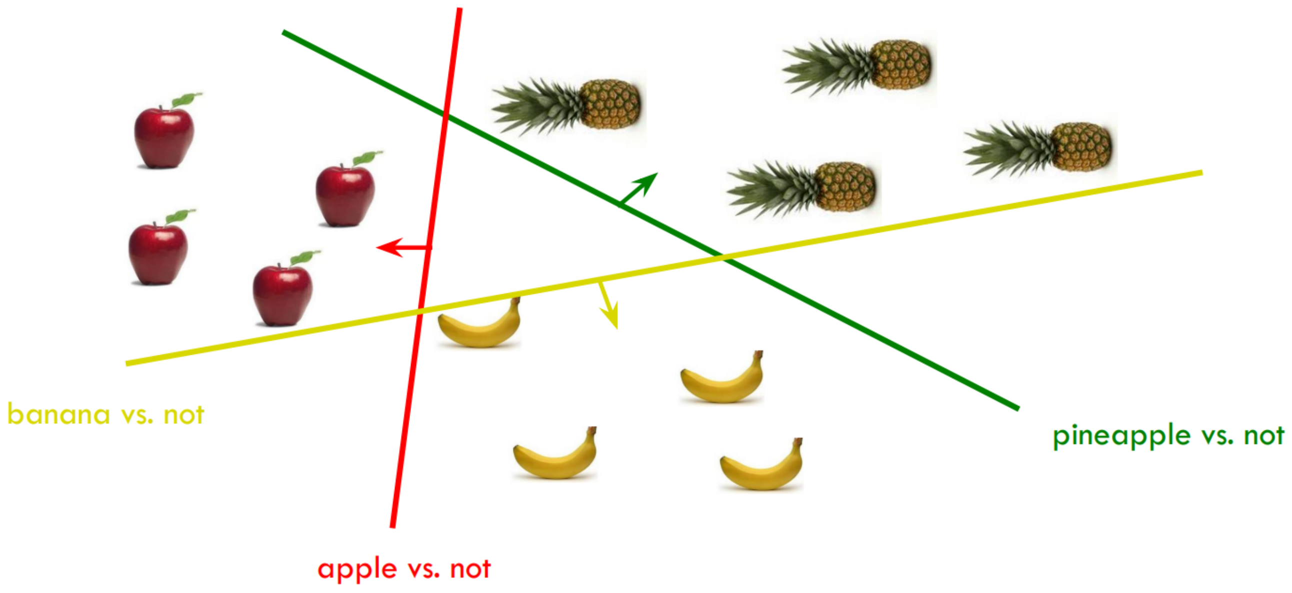
The sampling order can have profound effects on the outcome as seen in the image below, so when we say `for each training example (f_1, f_2, ... , f_n, label):` what we should really do is `randomly sample one example (f_1, f_2, ... , f_n, label):` .



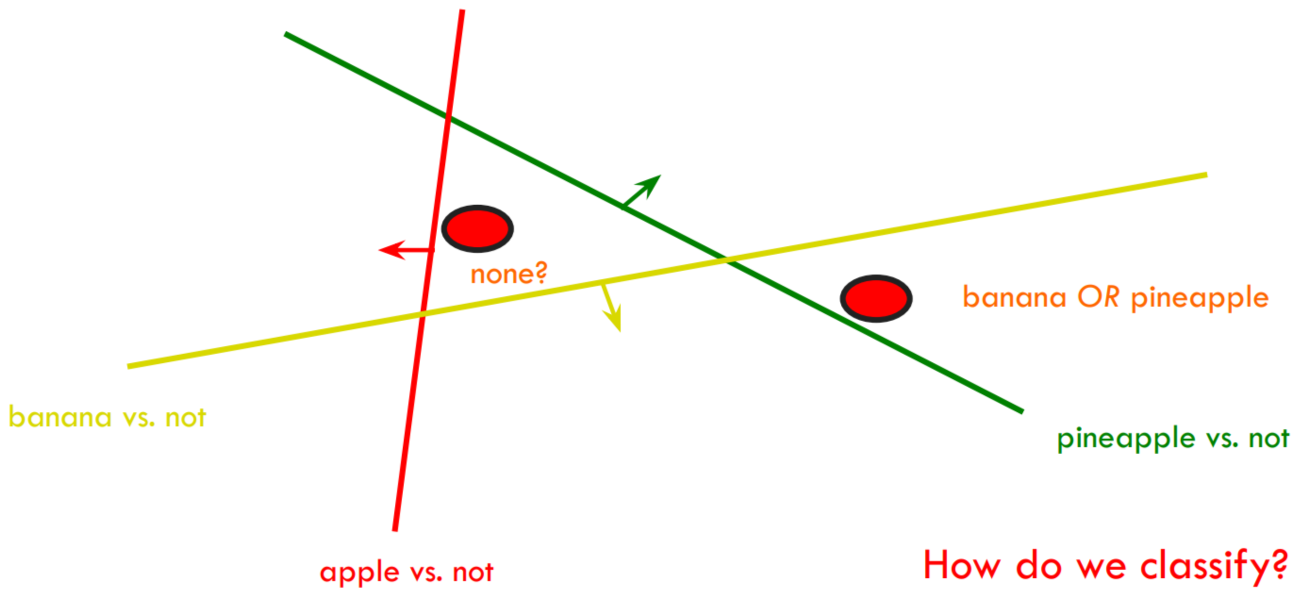
Using Perceptron for Multi-Class Classification

One Vs All[1]

The one vs all (#OVA) approach is based on a simple logical deduction, that is that if an example is part of one class it is not part of any of the others. The technique suggests defining a binary problem for each label L such that all examples with label L are *positive* and all other examples are *negative*. Effectively, we learn L different classification models.



This may cause us some problems, however, as there are some "dead" points in the defined space. For example:



Example

In this case we may suggest that the leftmost example in the above image is closest to being an apple, and therefore classify it as such. For the other example, we may see that while it is on the positive side of both the pineapple and banana lines, it is much further from the banana line and therefore is more likely to be a banana.

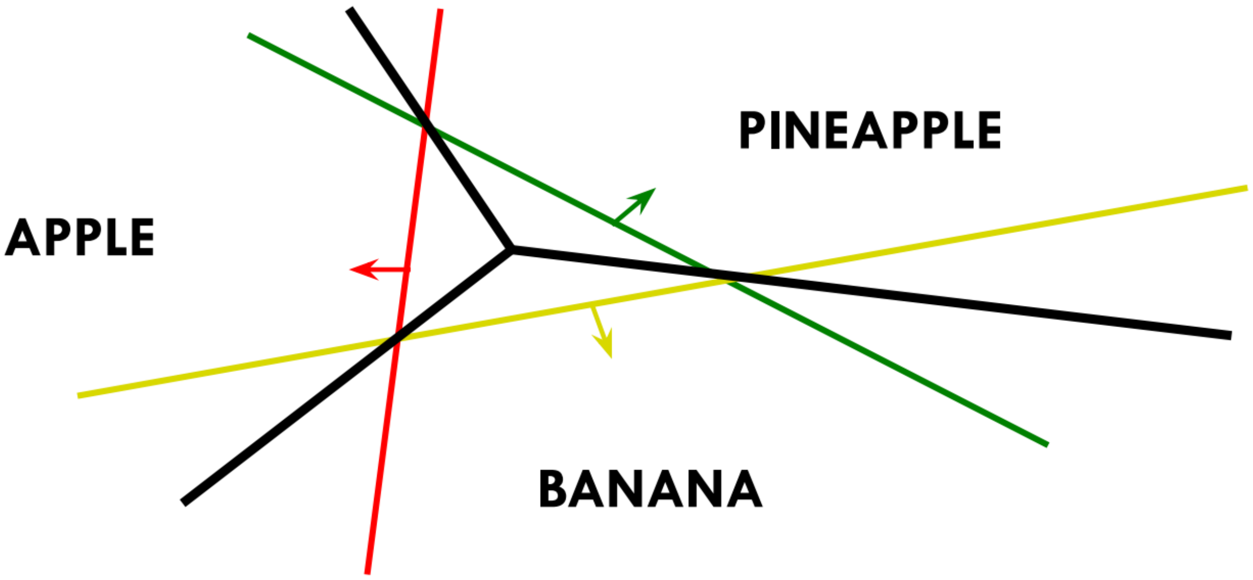
How do we formalise this?

If the classifier doesn't provide a measure of confidence and there's ambiguity[1], we can do nothing more than just picking one of the possible classes at random.
Generally, though, classifiers do provide some measure of confidence. In this case, in the presence of ambiguity, we should pick the *most confident positive*. If the classifier does not believe the example should be part of any of the available classes, we pick the *least confident negative*.

---

1. an example appears to be able to be part of more than one class ↩

This logic could be graphically represented with this decision boundary:



```
def oneVersusAllTrain(D_multiclass, BinaryTrain):
        for i in range(1, =K):
                D_bin = relabel D_multiclass so class i is positive and (not i) is
negative
                f[i] = BinaryTrain(D_bin)
        return f[1..K]

def oneVersusAllTest(f[1..K], x^):
        score = [0] * K  # initialise K scores to 0
        for i in range(1, =K):
                y = f[i](x^)
                score[i] = score[i] + y
        return max(score)
```

All Vs All[2]

The all vs all (#AVA) approach could be likened to a sports tournament, where every team has to play against every other team one at a time. The team that wins the most matches is the best.

We end up training $\frac{K(K-1)}{2}$[3] classifiers. $F_{i,\,j} \mid 1 \leq i < j \leq K$ is the classifier that pits class i against class j. When an example arrives we evaluate it on every $F_{i,\,j}$ classifier. Each classifier takes class i as the positive class and class j as the negative class, meaning that when $F_{i,\,j}$ predicts positive class i gets a vote and when it predicts negative class j gets a vote. After running all $\frac{K(K-1)}{2}$ classifiers, the class with the most votes wins.

To classify an example x we have two options:

- Take a majority vote
- Take a weighted vote based on confidence

One Vs All vs All Vs All[4]

Training time

#AVA learns more classifiers, but they're trained on much smaller datasets. Therefore, it is generally faster if the labels are equally balanced.

Testing time

#OVA has less classifiers, and therefore is generally faster.

Error

- #AVA trains on more balanced datasets
- #OVA tests with less classifiers and therefore has less chances for errors

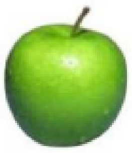
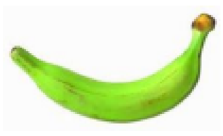
Summary

The most common approach to take is that of #OVA . Otherwise, a classifier that allows for multiple labels is usually chosen, such as #decision-trees or #K-NN which both work reasonably well. There are other, more sophisticated, methods which work better.

Evaluation

There are two approaches we can take:

- #Microaveraging
  - Average over examples
- #Macroaveraging
  - Calculate evaluation score[5] for each label, then average over the labels.

	label	prediction
	apple	orange
	orange	orange
	apple	apple
	banana	pineapple
	banana	banana
	pineapple	pineapple

In this example, the accuracy scores are significantly different. When using #microaveraging we clearly get $\frac{4}{6}$, but when using #macroaveraging the story is quite different. We classify $\frac{1}{2}$ of the apples correctly, $\frac{1}{1}$ of the oranges correctly, $\frac{1}{2}$ of the bananas correctly, and $\frac{1}{1}$ of the pineapples correctly. Taking the average of these we get: $\frac{\frac{1}{2}+\frac{1}{1}+\frac{1}{2}+\frac{1}{1}}{4} = \frac{3}{4}$

---

1. #OVA ↩
2. #AVA ↩
3. the number of distinct pairs we can make with K different classes ↩
4. OVA vs AVA ↩
5. accuracy ↩