Algorithm 1 AveragedPerceptronTrain($D, MaxIter$)

1: function AveragedPerceptronTrain($D, MaxIter$)
2: $w \leftarrow (0, 0, \ldots, 0)$, $b \leftarrow 0$
3: $u \leftarrow (0, 0, \ldots, 0)$, $\beta \leftarrow 0$
4: $c \leftarrow 1$
5: for $iter \leftarrow 1$ to $MaxIter$ do
6:     for $(x, y) \in D$ do
7:         if $y(wx + b) \leq 0$ then
8:             $w \leftarrow w + yx$
9:             $b \leftarrow b + y$
10:            $u \leftarrow u + yc x$
11:            $\beta \leftarrow \beta + yc$
12:         end if
13:     $c \leftarrow c + 1$
14:     end for
15: end for
16: return $w - \frac{1}{c} u, b - \frac{1}{c} \beta$
17: end function

Remember our learning procedure for averaged perceptron (shown in Algorithm 1).

Note that in this procedure, our scan over the training data with different epochs naturally
defines a sequence of the training data examples. We will call it the data sequence and denote
it as $T = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ be this data sequence for simplicity. Here $N$ is the
number of the examples in the data sequence and basically $N = |D| \times MaxIter$ ($|D|$ is the
number of examples in our training data $D$).

Also, remember the prediction rule for averaged perceptron:

$$\hat{y} = \text{sign} \left( \sum_{k=1}^{K} c^{(k)} w^{(k)} \cdot \hat{x} + \sum_{k=1}^{K} c^{(k)} h^{(k)} \right)$$

In averaged perceptron, we compute a weighted sum $S$ of the weight vectors $w^{(k)}$ that
we encounter during the our scan over the data sequence $T$ (we only talk about the weight
vectors here, but the argument extent naturally to the bias). The weight for each weight vector \( w^{(k)} \) in the weighted sum \( S \) is based on the survival time \( c^{(k)} \) of that weight vector in the data sequence (i.e., \( c^k \) is the ratio (over the entire sequence \( T \)) of the examples encountered right after \( w^k \) is produced and before \( w^k \) is replaced by \( w^{k+1} \) – the examples correctly predicted by \( w^{(k)} \):

\[
S = \sum_{k=1}^{K} c^{(k)} w^{(k)}
\]  

(1)

The goal of this note is to show that the Algorithm 1 is actually computing \( S \) (i.e., the returned value of \( w - \frac{1}{c} u \) is equal to \( S \)).

**Proof Sketch:**

First, the \( K \) variable in \( \hat{j} \) and \( S \) implies that we have \( K \) weight vectors along the scan over the data sequence. Note that based on the training procedure, we will only produce a new weight vector at an example in the sequence when the current weight vector cannot correctly classify that example. For convenience, let \( i_1, \ldots, i_K \) be the indexes of the examples in the data sequence for which we need to compute a new vector weight. Basically, we have \( i_1 < i_2 < \ldots < i_K \leq N \) and the weight vector produced at the example indexed at \( i_k \) (i.e., the example \( (x_{i_k}, y_{i_k}) \)) is \( w^{(k)} \) (due to the misclassification of \( w^{k-1} \) for \( x_{i_k} \)) (for all \( 1 \leq k \leq K \)).

Also, let \( i_0 = 0, i_{K+1} = N \) and \( w^0 = 0 \) for convenience.

With these notations, the survival time \( c^{(k)} \) for \( w^{(k)} \) can be computed by (i.e., the portions of examples between \( w^{(k)} \) and \( w^{(k+1)} \) over the entire sequence \( T \)):

\[
c^{(k)} = \frac{i_{k+1} - i_k}{N} \forall 1 \leq k \leq K
\]  

(2)

Also, based on the update rule of the training procedure, we can write \( w^{(k)} \) as:

\[
w^{(k)} = w^{(k-1)} + y_{i_k} x_{i_k} \forall 1 \leq k \leq K
\]  

(3)

By extending this equation, we have:

\[
w^{(k)} = w^{(k-1)} + y_{i_k} x_{i_k} = w^{(k-2)} + y_{i_{k-1}} x_{i_{k-1}} + y_{i_k} x_{i_k} = \ldots = w^{(0)} + y_{i_1} x_{i_1} + \ldots + y_{i_k} x_{i_k}
\]  

(4)

In other words, we have \( w^{(0)} = 0 \):

\[
w^{(k)} = \sum_{j=1}^{k} y_{i_j} x_{i_j} \forall 1 \leq k \leq K
\]  

(5)

Now, plugging Equations 2 and 5 to Equation 1, we obtain:

\[
S = \sum_{k=1}^{K} \frac{i_{k+1} - i_k}{N} \sum_{j=1}^{k} y_{i_j} x_{i_j}
\]  

(6)

Among the terms over \( k \) of \( S \) (i.e., \( \frac{i_{k+1} - i_k}{N} \sum_{j=1}^{k-1} y_{i_j} x_{i_j} \)), we note that \( y_{i_j} x_{i_j} \) only appears in the terms where \( k \geq j \). Also, there are \( K \) possible terms of the type \( y_{i_j} x_{i_j} \) with \( j \) ranging
from 1 to $K$ in $S$. Consequently, by grouping the terms of the $y_{ij}x_{ij}$ together, we can rewrite $S$ as follow:

$$S = \sum_{j=1}^{K} y_{ij}x_{ij} \sum_{k=j}^{K} \frac{i_{k+1} - i_k}{N} = \frac{1}{N} \sum_{j=1}^{K} y_{ij}x_{ij} \sum_{k=j}^{K} (i_{k+1} - i_k)$$ \hspace{1cm} (7)

Due to the cancellation, we have: $\sum_{k=j}^{K} (i_{k+1} - i_k) = i_{K+1} - i_j = N - i_j$, leading to:

$$S = \frac{1}{N} \sum_{j=1}^{K} y_{ij}x_{ij}(N - i_j) = \sum_{j=1}^{K} y_{ij}x_{ij} - \frac{1}{N} \sum_{j=1}^{K} y_{ij}i_jx_{ij}$$ \hspace{1cm} (8)

Now, consider the training procedure in Algorithm 1 again. We can see that the final value of the variable $w$ would involve an accumulation of the quantities $y_t x_t$ where $t$ is the index of one of the examples in $T$ for which we need to compute a new value or update the value for $w$ (i.e., $t \in \{i_1, i_2, \ldots, i_K\}$). In other words, the final value for $w$ is:

$$w = \sum_{j=1}^{K} y_{ij}x_{ij}$$ \hspace{1cm} (9)

Similarly, the final value of the variable $u$ would accumulate the quantities $y_t x_t$ for $t \in \{i_1, i_2, \ldots, i_K\}$ as the counter variable $c$ is essentially the index of the current example in $T$. Thus, the final value of $u$ is:

$$u = \sum_{j=1}^{K} y_{ij}i_jx_{ij}$$ \hspace{1cm} (10)

and the final value of $c$ is $c = N$.

Consequently, combining everything, the returned (or final) value for $w - \frac{1}{c} u$ is:

$$w - \frac{1}{c} u = \sum_{j=1}^{K} y_{ij}x_{ij} - \frac{1}{N} \sum_{j=1}^{K} y_{ij}i_jx_{ij}$$ \hspace{1cm} (11)

This is exactly the value for $S$ we show above and completes the proof.