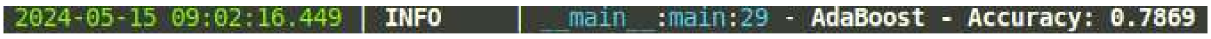
NYCU Pattern Recognition, Homework 3

312551810, Alexandre Pauly

Part. 1, Coding (60%):

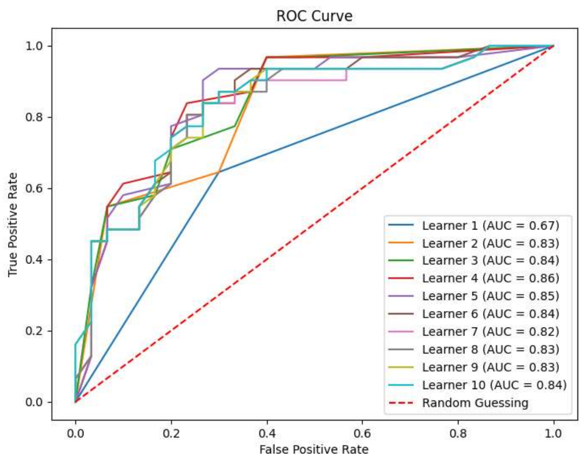
(20%) Adaboost

1. (10%) Show your accuracy of the testing data (n_estimators = 10)

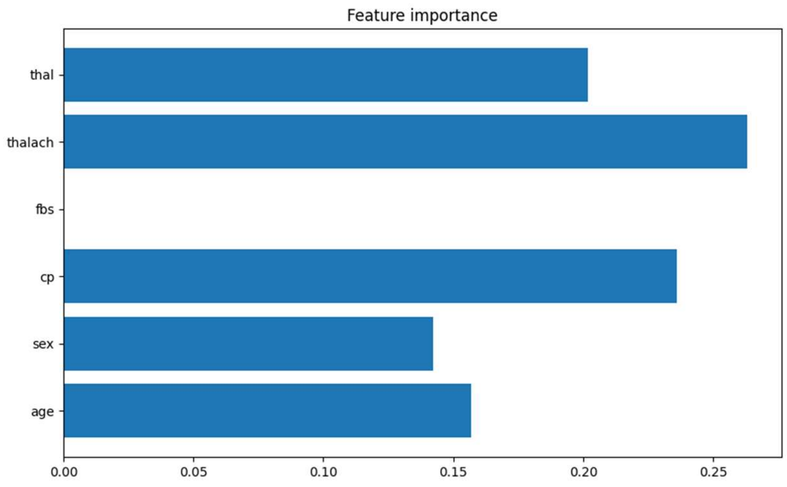


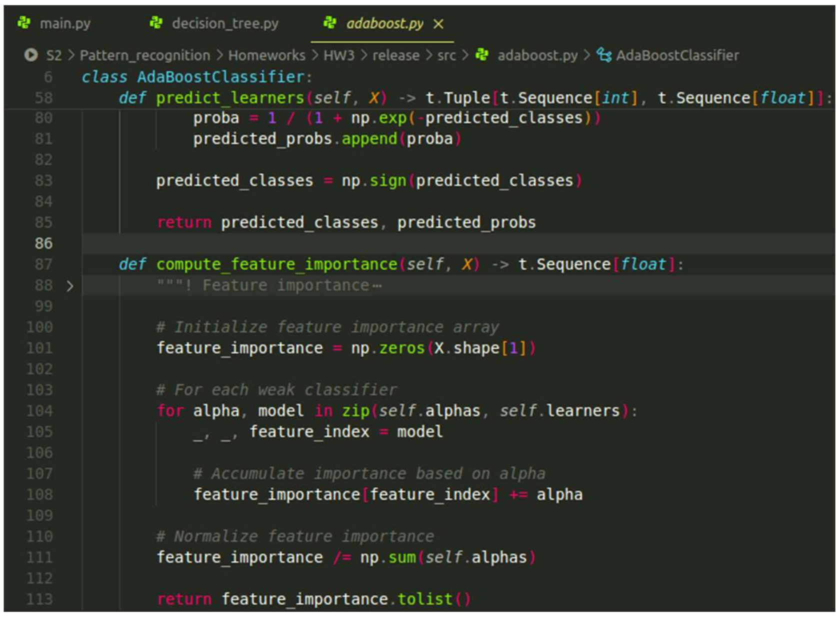
```
2024-05-15 09:02:16.449 | INFO     | __main__:main:29 - AdaBoost - Accuracy: 0.7869
```

2. (5%) Plot the AUC curves of <u>each</u> weak classifier.



3. (5%) Plot the feature importance of the AdaBoost method. Also, you should snapshot the implementation to calculate the feature importance.

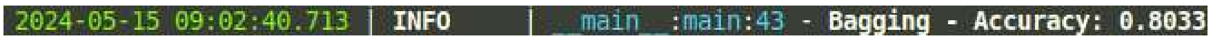




```
main.py    decision_tree.py    adaboost.py ×
S2 > Pattern_recognition > Homeworks > HW3 > release > src > adaboost.py > AdaBoostClassifier
class AdaBoostClassifier:
    def predict_learners(self, X) -> t.Tuple[t.Sequence[int], t.Sequence[float]]:
            proba = 1 / (1 + np.exp(-predicted_classes))
            predicted_probs.append(proba)

        predicted_classes = np.sign(predicted_classes)

        return predicted_classes, predicted_probs

    def compute_feature_importance(self, X) -> t.Sequence[float]:
        """! Feature importance⋯

        # Initialize feature importance array
        feature_importance = np.zeros(X.shape[1])

        # For each weak classifier
        for alpha, model in zip(self.alphas, self.learners):
            _, _, feature_index = model

            # Accumulate importance based on alpha
            feature_importance[feature_index] += alpha

        # Normalize feature importance
        feature_importance /= np.sum(self.alphas)

        return feature_importance.tolist()
```

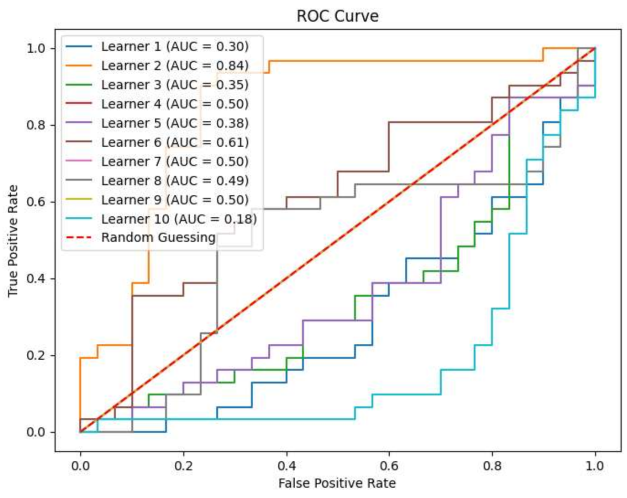
(20%) Bagging

4. (10%) Show your accuracy of the testing data with 10 estimators. (n_estimators=10)



```
2024-05-15 09:02:40.713 | INFO     | __main__:main:43 - Bagging - Accuracy: 0.8033
```

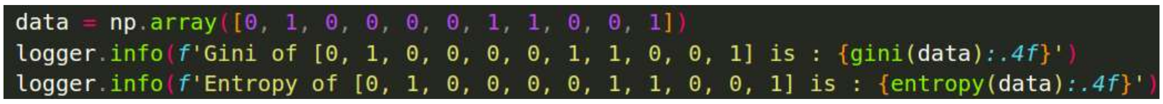
5. (5%) Plot the AUC curves of each weak classifier.



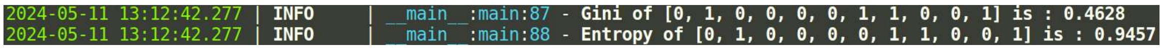
6. (5%) Plot the feature importance of the Bagging method. Also, you should snapshot the implementation to calculate the feature importance.

(15%) Decision Tree

7. (5%) Compute the gini index and the entropy of the array [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1].

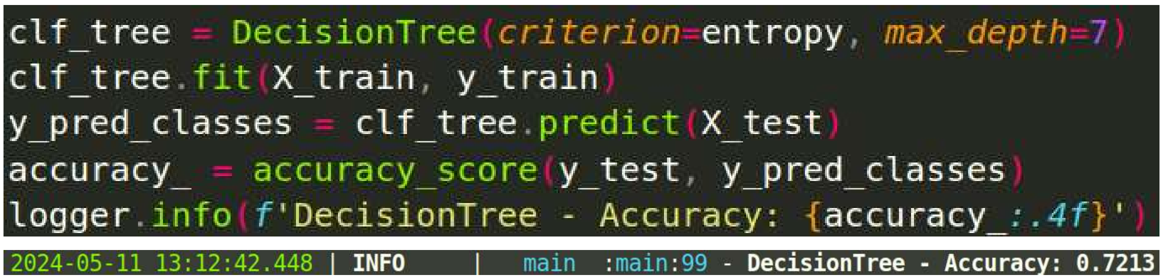


```
data = np.array([0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1])
logger.info(f'Gini of [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1] is : {gini(data):.4f}')
logger.info(f'Entropy of [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1] is : {entropy(data):.4f}')
```



```
2024-05-11 13:12:42.277 | INFO     | __main__:main:87 - Gini of [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1] is : 0.4628
2024-05-11 13:12:42.277 | INFO     | __main__:main:88 - Entropy of [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1] is : 0.9457
```

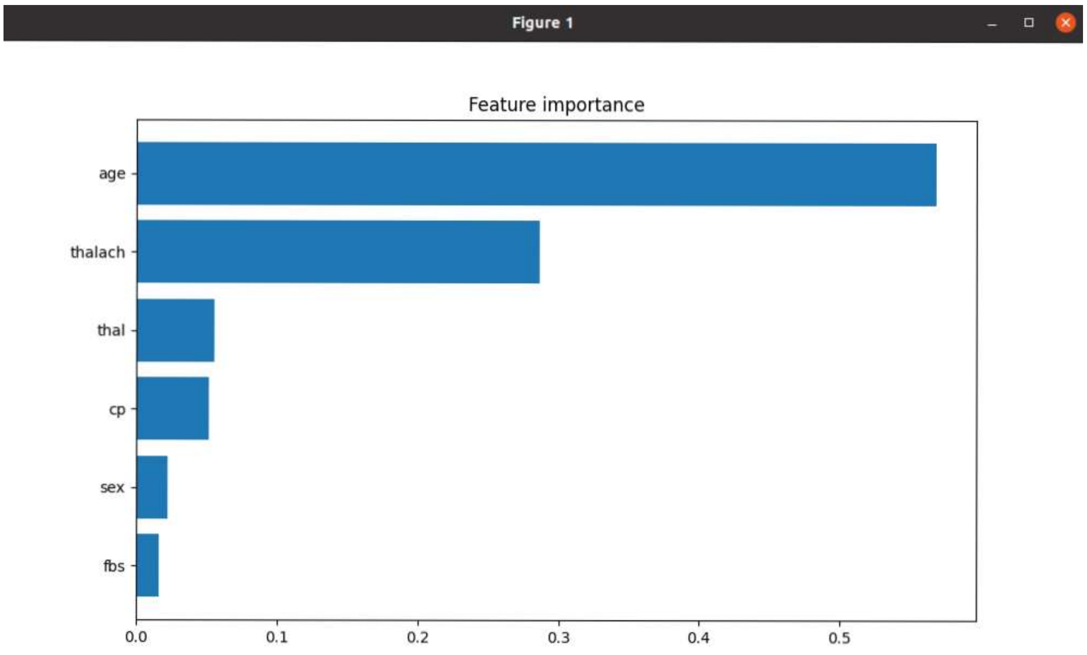
8. (5%) Show your accuracy of the testing data with a max-depth = 7

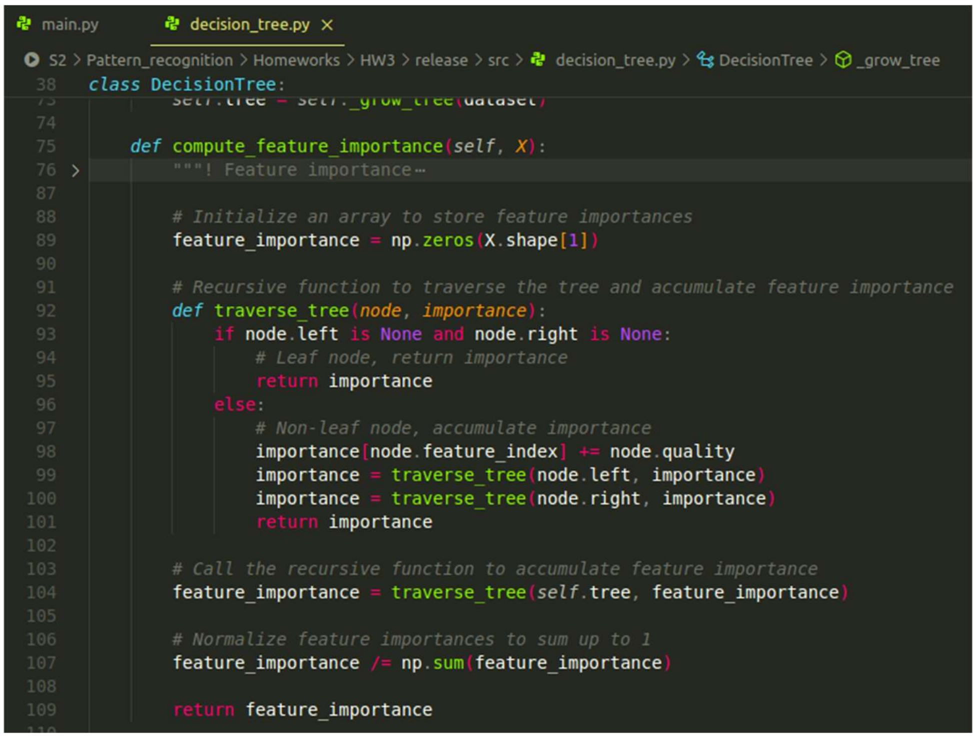


```
clf_tree = DecisionTree(criterion=entropy, max_depth=7)
clf_tree.fit(X_train, y_train)
y_pred_classes = clf_tree.predict(X_test)
accuracy_ = accuracy_score(y_test, y_pred_classes)
logger.info(f'DecisionTree - Accuracy: {accuracy_:.4f}')
```

```
2024-05-11 13:12:42.448 | INFO     | __main__:main:99 - DecisionTree - Accuracy: 0.7213
```

9. (5%) Plot the feature importance of the decision tree.

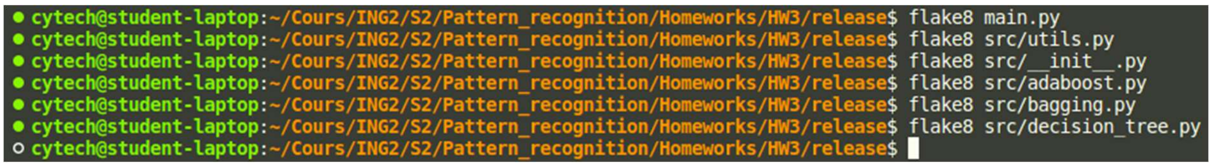




```
main.py    decision_tree.py
S2 > Pattern_recognition > Homeworks > HW3 > release > src > decision_tree.py > DecisionTree > _grow_tree
class DecisionTree:
        self.tree = self._grow_tree(dataset)

    def compute_feature_importance(self, X):
        """! Feature importance…

        # Initialize an array to store feature importances
        feature_importance = np.zeros(X.shape[1])

        # Recursive function to traverse the tree and accumulate feature importance
        def traverse_tree(node, importance):
            if node.left is None and node.right is None:
                # Leaf node, return importance
                return importance
            else:
                # Non-leaf node, accumulate importance
                importance[node.feature_index] += node.quality
                importance = traverse_tree(node.left, importance)
                importance = traverse_tree(node.right, importance)
                return importance

        # Call the recursive function to accumulate feature importance
        feature_importance = traverse_tree(self.tree, feature_importance)

        # Normalize feature importances to sum up to 1
        feature_importance /= np.sum(feature_importance)

        return feature_importance
```

(5%) Code Linting

10. Show the snapshot of the flake8 linting result.



```
● cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$ flake8 main.py
● cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$ flake8 src/utils.py
● cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$ flake8 src/__init__.py
● cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$ flake8 src/adaboost.py
● cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$ flake8 src/bagging.py
● cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$ flake8 src/decision_tree.py
○ cytech@student-laptop:~/Cours/ING2/S2/Pattern_recognition/Homeworks/HW3/release$
```

Part. 2, Questions (40%):

1. (10%) We have three distinct binary classifiers, and our goal is to leverage them in creating an ensemble classifier through the majority voting strategy to make decisions. Assuming each individual binary classifier operates independently of the others with an accuracy of 60%, what would be the accuracy of the ensemble classifier?

Hard voting :

To determine the accuracy of the ensemble classifier using the majority voting strategy, we can use the binomial probability formula.

Let's denote:

- n as the number of binary classifiers (in this case, n=3),
- p as the accuracy of each individual binary classifier (in this case, p=0.6),
- k as the number of classifiers that predict correctly out of n.

The accuracy of the ensemble classifier using the majority voting strategy can be calculated using the binomial probability formula:

$$P(\text{k out of n}) = \binom{n}{k} \times p^k \times (1-p)^{n-k}$$

In this scenario, since we're using majority voting, the ensemble classifier will make the correct prediction if at least two out of the three binary classifiers predict correctly. So, we need to calculate:

$$P(k\geq 2 \text{ out of } 3) = P(k = 2) + P(k = 3)$$

Let's calculate this:

$$P(k = 2) = \binom{3}{2} \times 0.6^2 \times (1 - 0.6)^{3-2}$$

$$P(k = 3) = \binom{3}{3} \times 0.6^3 \times (1 - 0.6)^{3-3}$$

Hard voting is based on the majority voting decision among the classifiers. If only one classifier predicts correctly while the other two predict incorrectly, the ensemble still makes the correct prediction based on the majority vote. Therefore, we don't need to calculate the probability of exactly one classifier being correct because it's included in the probability of either two or all three classifiers being correct. It’s the reason why we compute only $P(k = 2)$ and $P(k = 3)$.

Then, we can sum these probabilities to find the accuracy of the ensemble classifier:

$$\text{Accuracy} = P(k \geq 2 \text{ out of } 3)$$

Let's calculate it step by step:

$$P(k = 2) = \binom{3}{2} \times 0.6^2 \times (1-0.6)^{3-2} = 3 \times 0.62 \times 0.4 = 0.432$$

$$P(k = 3) = \binom{3}{3} \times 0.6^3 \times (1-0.6)^{3-3} = 1 \times 0.63 \times 0.40 = 0.216$$

$$\text{Accuracy} = P(k\geq 2 \text{ out of } 3) = P(k = 2) + P(k = 3) = 0.432 + 0.216 = 0.648$$

So, the accuracy of the ensemble classifier using the majority voting strategy would be 64.8%.

Soft voting :

In soft voting, the final prediction is determined by averaging the probabilities predicted by individual classifiers and choosing the class with the highest average probability. Since each classifier predicts with a 60% accuracy, it means that they are correct with a probability of 0.6 and incorrect with a probability of 0.4.

When averaging probabilities, the correct class will have a higher average probability if more classifiers predict it correctly.

For the ensemble to predict correctly, at least 3 out of 3 classifiers must predict the correct class. The average probability of the correct class in that case would be higher.

So, the accuracy of the ensemble classifier using soft voting is 60%.

2. (15%) For the decision tree algorithm, we can use the “pruning” technique to avoid overfitting. Does the random forest algorithm also need pruning? Please explain in detail.

Random forests, by design, inherently mitigate overfitting to a considerable extent without the need for pruning, and this is one of their key advantages over individual decision trees.

Here's why random forests typically don't require pruning:

1. **Ensemble of Trees**: Random forests are composed of an ensemble of decision trees. Each tree in the forest is built using a subset of the training data and a subset of the features. These subsets are selected randomly, hence the name "random forests". By constructing multiple trees with different subsets of data and features, random forests effectively reduce the risk of overfitting compared to a single decision tree.
2. **Voting Mechanism**: In classification tasks, random forests aggregate predictions from multiple trees through a majority voting mechanism. For regression tasks, they aggregate predictions through averaging. This ensemble approach helps to smooth out individual tree predictions, making the overall model more robust and less prone to overfitting.
3. **Bootstrapping and Feature Randomization**: The use of bootstrapping (random sampling with replacement) for training each tree and feature randomization (selecting a random subset of features at each split) further contributes to the diversity among trees in the forest. This diversity reduces the likelihood of individual trees overfitting to the training data.
4. **Out-of-Bag (OOB) Error Estimation**: Random forests utilize out-of-bag samples, which are data points not included in the bootstrap sample used to train each tree. These samples can be used to estimate the generalization error of the model without the need for a separate validation set. Monitoring the OOB error during training helps in diagnosing overfitting and making adjustments if necessary.

While random forests are less prone to overfitting compared to individual decision trees, there are still scenarios where they can overfit, especially if the number of trees in the forest is very large relative to the size of the dataset, or if the dataset itself is noisy. In such cases, techniques like limiting the maximum depth of trees, increasing the minimum number of samples required to split a node, or reducing the number of features considered at each split can be applied, but these are not considered "pruning" in the traditional sense of decision trees. Instead, they are more like hyperparameter tuning to control the complexity of the forest.

3. (15%) Activation functions are core components of neural networks. They need to be differentiable to ensure backpropagation works correctly. Please calculate the derivatives of the following commonly used activation functions.
(For questions 1. and 2., consider the cases where $x > 0$ and $x \leq 0$)

1. f(x) = relu(x),	df(x)/dx = ?
2. f(x) = leaky_relu(x) with negative_slope=0.01,	df(x)/dx = ?
3. f(x) = sigmoid(x),	df(x)/dx = ?

4. f(x) = silu(x),	df(x)/dx = ?
5. f(x) = tanh(x),	df(x)/dx = ?

1. $f(x) = relu(x) = \begin{cases} 0 \; if \; x \leq 0 \\ x \; if x > 0 \end{cases} = \max(0, x) = x \times IndicatorFunction_{x>0}$

$$and \; \frac{df(x)}{dx} = \begin{cases} 0 \; if \; x < 0 \\ 1 \; if \; x > 0 \\ undefined \; if \; x = 0 \end{cases}$$

2. f(x) = leaky_relu(x) with negative_slope=0.01 = $\begin{cases} 0.01x \; if \; x \leq 0 \\ x \; if x > 0 \end{cases} = \max(0.01x, x)$

$$and \; \frac{df(x)}{dx} = \begin{cases} 0.01 \; if \; x < 0 \\ 1 \; if \; x > 0 \\ undefined \; if \; x = 0 \end{cases}$$

3. $f(x) = sigmoid(x) = \frac{1}{1+e^{-x}}$

$$and \; \frac{df(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2}$$
$$= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)$$
$$= sigmoid(x)(1 - sigmoid(x))$$

4. $f(x) = silu(x) = \frac{x}{1+e^{-x}}$

$$and \; \frac{df(x)}{dx} = \frac{1+e^{-x}+xe^{-x}}{(1+e^{-x})^2}$$

5. $f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$$and \; \frac{df(x)}{dx} = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$
$$= 1 - \frac{(e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} = 1 - (tanh(x))^2$$