UDACITY

Return to Classroom

DISCUSS ON STUDENT HUB

Combine Signals for Enhanced Alpha

REVIEW

CODE REVIEW

HISTORY

Requires Changes

1 specification requires changes

Great job, you have almost got this one!

There is just one issue in this challenging project that has caught you:

- You have forgotten to send in a different value for starting index when calling non_overlapping_sample in your **non_overlapping_estimators** function.

Please read through the comments and suggestions and make the appropriate corrections and then re-submit.

Good luck on your re-submission and on your next and final project in the nanodegree!!

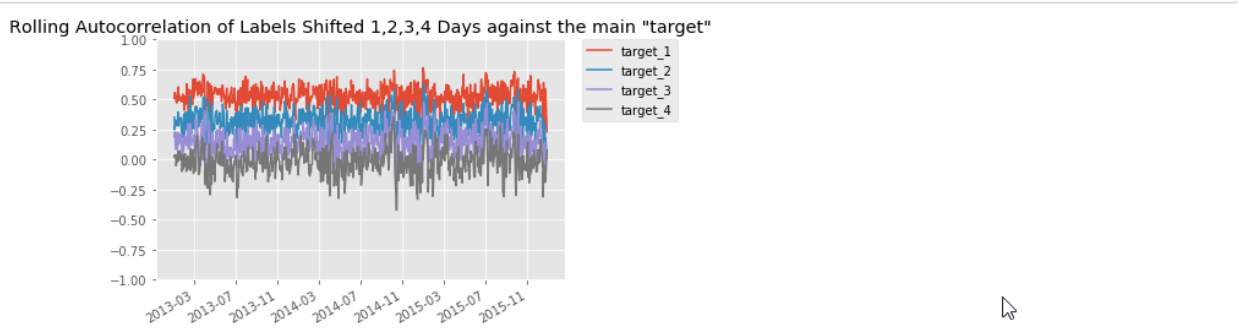
It has been a pleasure reviewing your project.

Features and Labels

Describe the relationship between the shifted labels.

You are correct!

- There is high correlation between next days returns.
- You can learn more about **IID** and **Autocorrelation** at these urls:
  - https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/
  - https://www.youtube.com/watch?v=MlqsXYuvClw
  - https://stats.stackexchange.com/questions/213464/on-the-importance-of-the-i-i-d-assumption-in-statistical-learning




```
# Notes from Pandas:
# Indexing will work even if the data are not sorted, but will be rather inefficient (and show a PerformanceWarning).
# It will also return a copy of the data rather than a view.
all_factors.sort_index(inplace=True)
```

```
# save final factors
# all_factors.to_csv('all_factors_final.csv')
```

Question: What do you observe in the rolling autocorrelation of labels shifted?

#TODO: Put Answer In this Cell

The rolling autocorrelations increase as shifted days are larger. The autocorrelations are positive with target_1 (4d shifted), target_2 (3d shifted), and target_3 (2d shifted) which imply the future 5d returns moving to the same direction as the past 5d returns. They affect more by past returns that have closer dates to the current date.

target_4 (1d shifted) is around 0, so it does not mean much.

Correctly implement the `train_valid_test_split` function.

You have correctly implemented the splitting of the data between train, validation, and test

- **and** you have not split any day between multiple datasets! Well done!

We see in this screenshot, that you are making sure not to split a day between multiple datasets!!

- see how the dates are **different** between the end of Train and the beginning of Valid?

```
print("Last 10 of Train:")
print(X_train.iloc[-10:]['Mean_Reversion_Sector_Neutral_Smoothed'])
print('*' * 80)
print("First 10 of Valid:")
print(X_valid.iloc[:10]['Mean_Reversion_Sector_Neutral_Smoothed'])

```

```
Last 10 of Train:
2014-10-21 00:00:00+00:00  Equity(481 [XL])     -0.25002462
                           Equity(482 [XLNX])   -0.21378917
                           Equity(483 [XOM])    -1.57624215
                           Equity(484 [XRAY])    0.86602730
                           Equity(485 [XRX])     0.01811773
                           Equity(486 [XYL])     1.11967546
                           Equity(487 [YUM])    -0.83703894
                           Equity(488 [ZBH])    -0.37322515
                           Equity(489 [ZION])    0.51091987
                           Equity(490 [ZTS])    -1.25737018
Name: Mean_Reversion_Sector_Neutral_Smoothed, dtype: float64
********************************************************************************
First 10 of Valid:
2014-10-22 00:00:00+00:00  Equity(0 [A])        0.05785569
                           Equity(1 [AAL])      1.50424782
                           Equity(2 [AAP])     -0.88953116
                           Equity(3 [AAPL])    -1.35237665
                           Equity(4 [ABBV])    -0.44114960
                           Equity(5 [ABC])     -0.67257234
                           Equity(6 [ABT])      0.52793313
                           Equity(7 [ACN])     -1.04863430
                           Equity(8 [ADBE])     0.79551567
                           Equity(9 [ADI])      0.61471666
Name: Mean_Reversion_Sector_Neutral_Smoothed, dtype: float64
```

```
    valid_size : float
        The proportion of the data used for the validation dataset
    test_size : float
        The proportion of the data used for the test dataset

    Returns
    -------
    x_train : DataFrame
        The train input samples
    x_valid : DataFrame
        The validation input samples
    x_test : DataFrame
        The test input samples
    y_train : Pandas Series
        The train target values
    y_valid : Pandas Series
        The validation target values
    y_test : Pandas Series
        The test target values
    """
    assert train_size >= 0 and train_size <= 1.0
    assert valid_size >= 0 and valid_size <= 1.0
    assert test_size >= 0 and test_size <= 1.0
    assert train_size + valid_size + test_size == 1.0

    # TODO: Implement
    days_indices = all_x.index.levels[0]
    n_days = len(days_indices)
    train_cutoff = int(train_size * n_days)
    valid_cutoff = int((train_size + valid_size) * n_days)

    days_train_indices, days_valid_indices, days_test_indices = days_indices[:train_cutoff], days_indices[train_cutoff:val

    x_train, x_valid, x_test = all_x.loc[days_train_indices[0]:days_train_indices[-1]], all_x.loc[days_valid_indices[0]:da
    y_train, y_valid, y_test = all_y.loc[days_train_indices[0]:days_train_indices[-1]], all_y.loc[days_valid_indices[0]:da

    return x_train, x_valid, x_test, y_train, y_valid, y_test
```

```
project_tests.test_train_valid_test_split(train_valid_test_split)
```

```
Tests Passed
```

Random Forests

Describe why dispersion_20d has the highest feature importance, when the first split is on the Momentum_1YR feature.

Exactly correct!

- Well done!
- When determining the first split, we desire to maximize the information gain and at the time of the first split, **Momentum_1YR** is determined to be the best. After multiple splits have been made, **dispersion_20d** results as the highest importance.

Question: Why does dispersion_20d have the highest feature importance while the first split is on the Momentum_1YR feature?

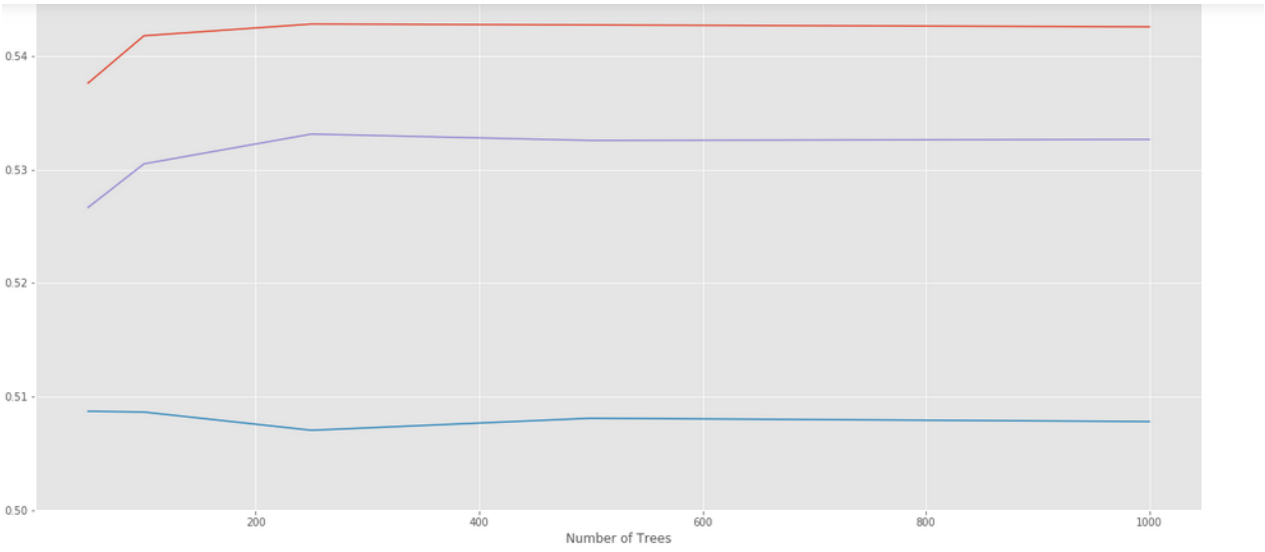
#TODO: Put Answer In this Cell

`dispersion_20d` was used 3 times to split the tree while `Momentum_1YR` was used once. Also the entropies from 2 chidl nodes of `Momentum_1YR` remain `1.0` as their parent while entropies from the child nodes of `dispersion_20d` were down a bit (`0.9`) so `dispersion_20d` is more important than `Momentum_1YR`

Describe how the accuracy changes over time and what indicates the model is overfitting or underfitting.

Yes!

- After a number of increases in trees (which implies more complex models), the accuracy stops getting better.
- Additionally, the large difference between train accuracy and validation accuracy (with train accuracy much higher) indicates **overfitting**.



Better explanation of OOB:

What is Out of Bag (OOB) score in Random Forest?

Question:

1. What do you observe with the accuracy vs tree size graph?
2. Does the graph indicate the model is overfitting or underfitting? Describe how it indicates this.

#TODO: Put Answer In this Cell

1. For training and OOB sub datasets, the accuracy increases steady when the number of trees goes from 50 to 100. It continues to grow slower up to 250 trees then stop. For validation dataset, its accurary goes down a bit from 100 trees to 250 trees, but overall, it does not change much. Very steadily the same.
2. The model is overfitting because of the large gap between training/oob and validation. Also training and validation accuracies never converged

Overlapping Samples

Correctly implement the `non_overlapping_samples` function.

You have correctly implemented the non_overlapping_samples function.

- Key to your solution was to use the split from start_i to the end incrementing by n_skip_samples+1 [start_i::n_skip_samples+1]

```
def non_overlapping_samples(x, y, n_skip_samples, start_i=0):
    """
    Get the non overlapping samples.

    Parameters
    ----------
    x : DataFrame
        The input samples
    y : Pandas Series
        The target values
    n_skip_samples : int
        The number of samples to skip
    start_i : int
        The starting index to use for the data

    Returns
    -------
    non_overlapping_x : 2 dimensional Ndarray
        The non overlapping input samples
    non_overlapping_y : 1 dimensional Ndarray
        The non overlapping target values
    """
    assert len(x.shape) == 2
    assert len(y.shape) == 1

    # TODO: Implement
    non_overlapping_x = x.loc[x.index.levels[0][start_i::n_skip_samples+1].tolist()]
    non_overlapping_y = y.loc[y.index.levels[0][start_i::n_skip_samples+1].tolist()]

    return non_overlapping_x, non_overlapping_y
```

```
project_tests.test_non_overlapping_samples(non_overlapping_samples)
```

```
Tests Passed
```

Correctly implement the `bagging_classifier` function.

Nicely done:

- You are properly creating an instance of **BaggingClassifier** using the passed in parameters!

```
    n_estimators : int
        The number of base estimators in the ensemble
    max_samples : float
        The proportion of input samples drawn from when training each base estimator
    max_features : float
        The proportion of input sample features drawn from when training each base estimator
    parameters : dict
        Parameters to use in building the bagging classifier
        It should contain the following parameters:
            criterion
            min_samples_leaf
            oob_score
            n_jobs
            random_state

    Returns
    -------
    bagging_clf : Scikit-Learn BaggingClassifier
        The bagging classifier
    """

    required_parameters = {'criterion', 'min_samples_leaf', 'oob_score', 'n_jobs', 'random_state'}
    assert not required_parameters - set(parameters.keys())

    # TODO: Implement
    base_estimator = DecisionTreeClassifier(
        criterion=parameters['criterion'],
        min_samples_leaf=parameters['min_samples_leaf']
    )

    bagging_clf = BaggingClassifier(
        base_estimator=base_estimator,
        n_estimators=n_estimators,
        max_samples=max_samples,
        max_features=max_features,
        oob_score=parameters['oob_score'],
        random_state=parameters['random_state'],
        n_jobs=parameters['n_jobs'])

    return bagging_clf
```

```
project_tests.test_bagging_classifier(bagging_classifier)
```

```
Tests Passed
```

Correctly implement the `calculate_oob_score` function.

Very concise and straight forward solution. Good Job!

- You are accurately averaging all the estimator's OOB scores!!

You could even simplify a little more using a list comprehension and numpy's mean function:

```
    return np.mean([clf.oob_score_ for clf in classifiers])
```

```python
def calculate_oob_score(classifiers):
    """
    Calculate the mean out-of-bag score from the classifiers.

    Parameters
    ----------
    classifiers : list of Scikit-Learn Classifiers
        The classifiers used to calculate the mean out-of-bag score

    Returns
    -------
    oob_score : float
        The mean out-of-bag score
    """

    # TODO: Implement
    scores = []
    for classifier in classifiers:
        scores.append(classifier.oob_score_)

    return np.mean(scores)
```

```python
project_tests.test_calculate_oob_score(calculate_oob_score)
```

```
Tests Passed
```

Correctly implement the `non_overlapping_estimators` function.

This is **almost** correct . . .

- you **are** attempting to create subsets of non-overlapping data by calling the **non_overlapping_samples** function and using those subsets to feed each classifier (rather than using the same data for each classifier).

However, you are **not** properly calling **non-overlapping_samples** -

- recall that **non_overlapping_samples** takes **4** parameters - you are only passing 3. The last parameter is the starting index to use when creating the samples it has a default value of **0** - each of your samples is identical and you are then sending identical data to each of the classifiers . . .

if we add some print statements inside your loop:

```
        print(f"Sample: {i+1}:")
        print(x_train)
        print('*' * 80)
```

```
        print(' ')
```

We can see how the data is identical each time:

```
project_tests.test_non_overlapping_estimators(non_overlapping_estimators)
```

```
Sample: 1:
                           test column 1  test column 2
2016-11-12 Equity(0 [A])                0             24
           Equity(1 [AAL])              1             25
           Equity(2 [AAP])              2             26
2016-11-16 Equity(0 [A])               12             36
           Equity(1 [AAL])             13             37
           Equity(2 [AAP])             14             38
********************************************************************************

Sample: 2:
                           test column 1  test column 2
2016-11-12 Equity(0 [A])                0             24
           Equity(1 [AAL])              1             25
           Equity(2 [AAP])              2             26
2016-11-16 Equity(0 [A])               12             36
           Equity(1 [AAL])             13             37
           Equity(2 [AAP])             14             38
********************************************************************************

Sample: 3:
                           test column 1  test column 2
2016-11-12 Equity(0 [A])                0             24
           Equity(1 [AAL])              1             25
           Equity(2 [AAP])              2             26
2016-11-16 Equity(0 [A])               12             36
           Equity(1 [AAL])             13             37
           Equity(2 [AAP])             14             38
********************************************************************************
```

Consider keeping an iterator variable to hold the iteration number and using that for your start_index in the call to **non_overlapping_samples**

Hint: Python has the **enumerate** function specifically designed for doing for X in Collection when you also need an index

https://stackoverflow.com/questions/522563/accessing-the-index-in-for-loops#522578

When properly implemented, you should see output that shows that each sample is different and non-overlapping:

```
Sample: 1:
                           test column 1  test column 2
2001-08-25 Equity(0 [A])               0             24
           Equity(1 [AAL])             1             25
           Equity(2 [AAP])             2             26
2001-08-29 Equity(0 [A])              12             36
           Equity(1 [AAL])            13             37
           Equity(2 [AAP])            14             38
********************************************************************************

Sample: 2:
                           test column 1  test column 2
2001-08-26 Equity(0 [A])               3             27
           Equity(1 [AAL])             4             28
           Equity(2 [AAP])             5             29
2001-08-30 Equity(0 [A])              15             39
           Equity(1 [AAL])            16             40
           Equity(2 [AAP])            17             41
********************************************************************************

Sample: 3:
                           test column 1  test column 2
2001-08-27 Equity(0 [A])               6             30
           Equity(1 [AAL])             7             31
           Equity(2 [AAP])             8             32
2001-08-31 Equity(0 [A])              18             42
           Equity(1 [AAL])            19             43
           Equity(2 [AAP])            20             44
********************************************************************************
```

You may find this post at Udacity's Knowledge Base - https://knowledge.udacity.com/questions/32823 quite helpful . . .

RESUBMIT

DOWNLOAD PROJECT