Boosting Methods

MIDIES DATE:

Section Overview:

- Boosting and Meta-learning
- AdaBoost (Adaptive Boosting) Theory
- Example of AdaBoost
- Gradient Boosting Theory
- Example of Gradient Boosting

2- Boosting Motivation and History

The concept of Boosting is not actually a machine learning algorithm, it is methodology applied to an existing machine learning algorithm, most commonly applied to decision tree.

- main formula for boosting

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

$$f_t(x) = \alpha_t h(t)$$

- Implies that a combination of estimators with an applied coefficient could act as an effective ensemble estimator.

M T W T F S DATE:

- Note h(x) can in theory be any machine learning algorithm (estimator/learner)
- Can an ensemble of weaker learners (very simple models) be a strong learner when combined?

- For decision tree models, we can use simple tree in place of h(x) and combine them with the coefficients on each model.

- The idea of gradient boosting originated from Leo Breiman when he observed that boosting can be interpreted as an optimized ~~forecast~~ algorithm on a cost function in publications in the late 1990s.
- Later on Jerome H Friedman and many others developed more explicit formulations of gradient boosting.

- Also in the late 1990s Yoav Freund and Robert Schapire developed the AdaBoost (Adaptive Boosting algorithm, which also combines weak learner in an ensemble to create a ~~strong~~ stronger model.

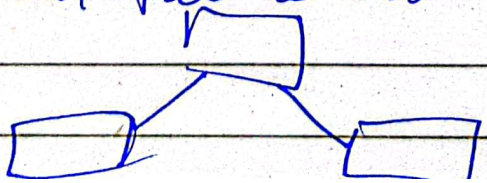
NOTES DATE:

- let is continue by focusing first on AdaBost and building an understanding of how to combine week learners to create a strong estimator.
- We will also explore why Decision Trees are so suited for boosting

3. AdaBoost Theory and Intuit

- AdaBoost (Adaptive Boosting) work by using an ensamble of weak learners and then cumbining them through the use of weighted sum
- AdaBoost adapts by wing previously created weak learners in order to adjust misclassified instances for the next created weak learner.

- What is a Weak learner?
  - A week learner is a model that is too simple to perform well on its own.
  - The weakest decision tree possible would be a stump | one node and two leaves -



M T W T F S DATE:

Slides have the necessary data
Check them

004 : Adaboost Coding part 1

Coding

```
df = pd.read_csv('.../Data/musrooms.csv')
df.head()

sns.countplot(data=df, x="class'
df.describe()

df.describe().transpose().reset_index().
feat =                    sort_values('unique')

sns.barplot(data=feat_u, x='index', y='unique')
plt.xticks(rotation=90);

X = df.drop('class', axis=1)
X.isnull().sum()
X = pd.getdummies(X, drop_first=True)
y = df['class']
```

M T W T F S DATE:

```
train_test_split(X, y                )
```

part 2
Adaboost Classier

```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(n_estimators=1)
model.fit(X_train, y_train)

# metrics

predictions = model.predic(X_test)
predictions
print(classification_Report)

model_feautures_importance.argmax()

X.colums[22]
                                  n="odor"
sns.countplot(data=df, hue="class")

len(X.colums)

for loop n in Range(1,96):
    model = AdaBoostClassifier(n_estimators=n)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    err = 1-Accuracy(ytest, ypreds)
    error_rates.append(err)

plt.plot(range(1,96), error_rates)
```

M T W T F S   DATE:

model

feats = pd.DataFrame(X.columns, data = model_importance)

Gradient Boosting Theory

Gradient boosting is very similar idea to AdaBoost where week learners are created in series in order to produce a strong ensemble ~~method~~ model

- Gradient Boosting make use of residual error for learning.

- Gradient boosting vs Adaboost:
  - Larger tree allowed in Gradient Boosting
  - Learning rate coefficient same for all weak learners.
  - Gradual series learning is based on training on the residual of previous model

DATE:

y	$\hat{y}$	e	$f1$	$F1 = \hat{y} + f1$
5000000	509,000	9000	8000	501,000
462000	509000	47000	50000	459000
565000	509000	56000	50000	559,000

- ~~Create new mode to predict the error~~
- update prediction using error prediction
- **Gradient Boosting process:-**

~~$F_m = F_{m-1} + f_m$~~

$F_m = F_{m-1} + (\text{learning rate} * f_m)$

- Create Initial model
- Train another model on error
- $e = y - f_0$
- Create new prediction

$F_1 = f_0 + \eta f_1$

- Repeat as needed

$F_m = f_{m-1} + \eta f_m$

- Note, for classification we can use the Logi as an error metric:

$\hat{y} = \log\left(\frac{\hat{p}}{1-\hat{p}}\right) \quad \hat{p} = \frac{1}{1+e^{-\hat{y}}}$

M T W T F S DATE:

- Note: the learning rate is not same for each new model in the series, it is not unique to each subsequent model (unlike AdaBoosts alpha coefficient).
- Gradient Boosting is fairly robust to overfitting, allowing for the number of estimators to be set high default (~100)

- Gradient Boosting Intuition:
  - we optimize the series of trees by learning on the residuals, forcing subsequents trees to attempt to correct for the error in the previous trees.

- The trade off is training time,
  - A learning rate is between 0-1 which mean a very low value would mean each subsequent tree has little "say", meaning more trees need to be created, causing a larger computation training time.

Gradient boosting coding

M T W T F S DATE:

Same as before for Adaboost

import
from sklearn.ensemble . GradientBoostingClassifier

param_grids = { 'n_estimators': [50, 100],
'learning_rate': [0.1, 0.05, 0.2],
'max_depth': [3, 4, 5] }

gb_model = GradientBoostingClassifier()
grid = GridSearchCV(gb_model,
param_grids)

grid.fit(x_train, y_train)

metrics