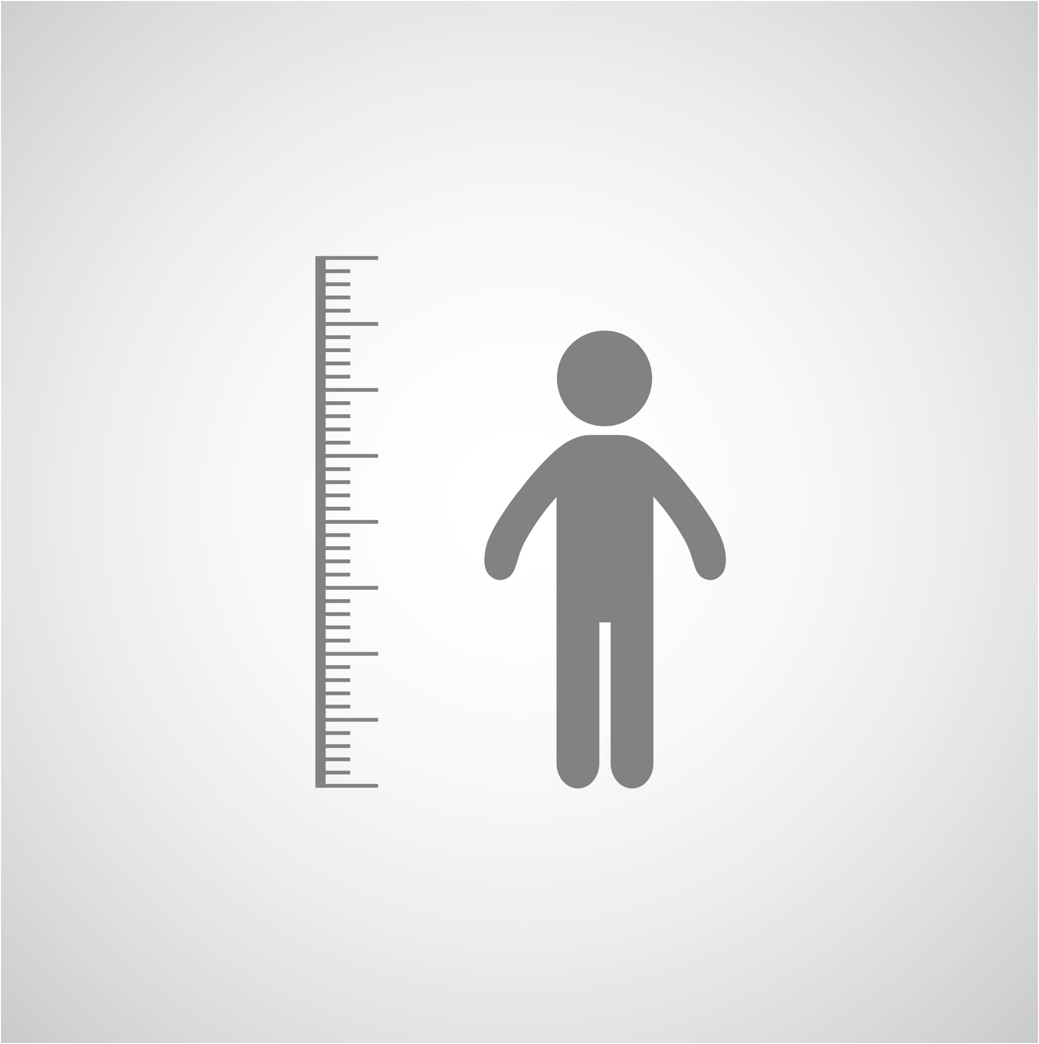
# Regression review

EXTREME GRADIENT BOOSTING WITH XGBOOST

Sergey Fogelson
VP of Analytics, Viacom

# Regression basics

- Outcome is real-valued



# Common regression metrics

- Root mean squared error (RMSE)
- Mean absolute error (MAE)

# Computing RMSE

| Actual | Predicted |
|---|---|
| 10 | 20 |
| 3 | 8 |
| 6 | 1 |

# Computing RMSE

| Actual | Predicted | Error |
|---|---|---|
| 10 | 20 | -10 |
| 3 | 8 | -5 |
| 6 | 1 | 5 |

# Computing RMSE

| Actual | Predicted | Error | Squared Error |
|---|---|---|---|
| 10 | 20 | -10 | 100 |
| 3 | 8 | -5 | 25 |
| 6 | 1 | 5 | 25 |

- Total Squared Error: 150
- Mean Squared Error: 50
- Root Mean Squared Error: 7.07

# Computing MAE

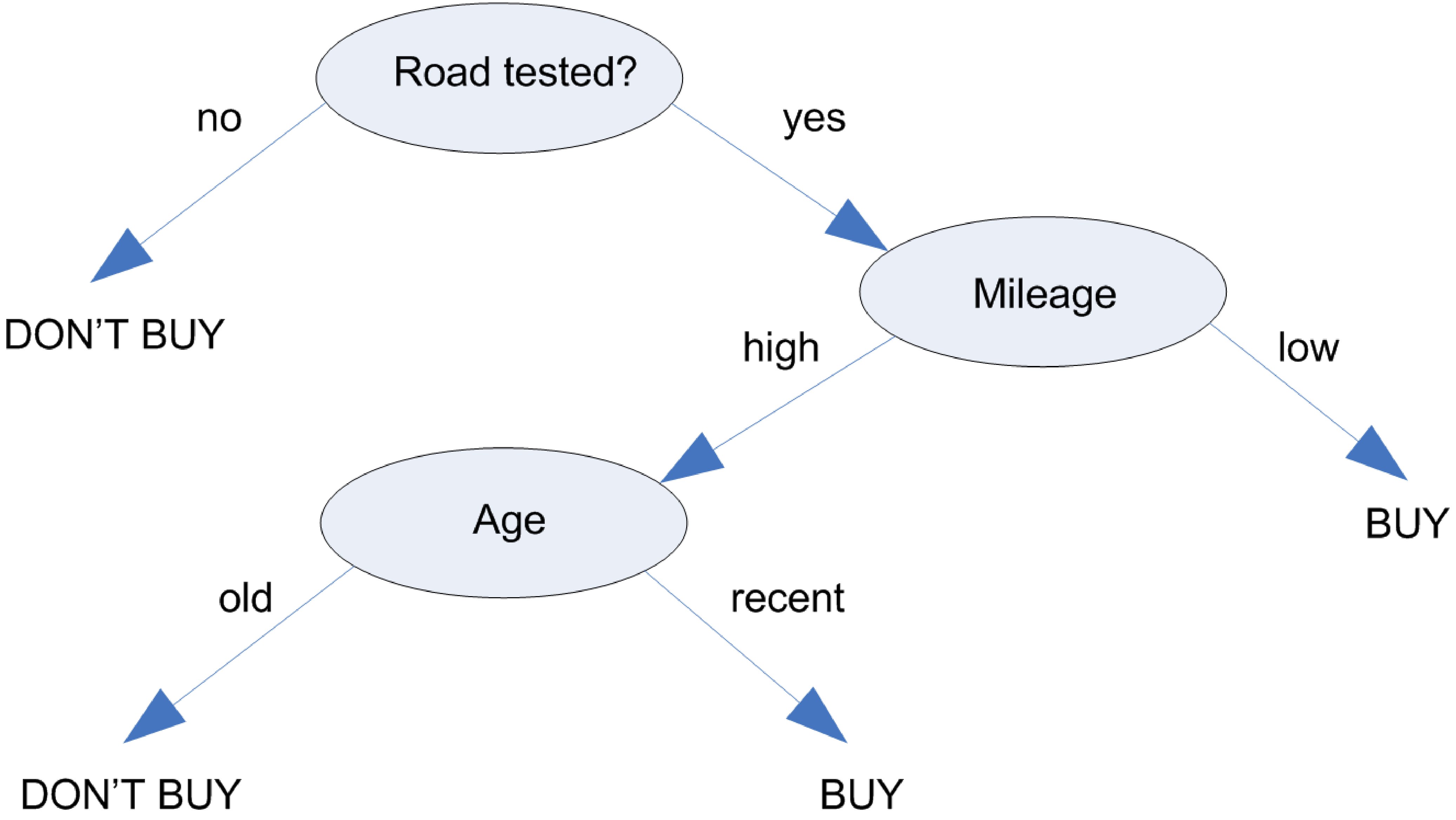
| Actual | Predicted | Error |
|---|---|---|
| 10 | 20 | -10 |
| 3 | 8 | -5 |
| 6 | 1 | 5 |

- Total Absolute Error: 20
- Mean Absolute Error: 6.67

# Common regression algorithms

- Linear regression
- Decision trees

# Algorithms for both regression and classification




1 https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.

# Let's practice!

EXTREME GRADIENT BOOSTING WITH XGBOOST

# Objective (loss) functions and base learners

EXTREME GRADIENT BOOSTING WITH XGBOOST

**Sergey Fogelson**
VP of Analytics, Viacom

# Objective Functions and Why We Use Them

- Quantifies how far off a prediction is from the actual result
- Measures the difference between estimated and true values for some collection of data
- Goal: Find the model that yields the minimum value of the loss function

# Common Loss Functions and XGBoost

- Loss function names in xgboost:
  - reg:linear - use for regression problems
  - reg:logistic - use for classification problems when you want just decision, not probability
  - binary:logistic - use when you want probability rather than just decision

# Base Learners and Why We Need Them

- XGBoost involves creating a meta-model that is composed of many individual models that combine to give a final prediction
- Individual models = base learners
- Want base learners that when combined create final prediction that is **non-linear**
- Each base learner should be good at distinguishing or predicting different parts of the dataset
- Two kinds of base learners: tree and linear

# Trees as Base Learners example: Scikit-learn API

```
In [1]: import xgboost as xgb
In [2]: import pandas as pd
In [3]: import numpy as np
In [4]: from sklearn.model_selection import train_test_split

In [5]: boston_data = pd.read_csv("boston_housing.csv")
In [6]: X, y = boston_data.iloc[:,:-1],boston_data.iloc[:,-1]

In [7]: X_train, X_test, y_train, y_test= train_test_split(X, y,
        test_size=0.2, random_state=123)
In [8]: xg_reg = xgb.XGBRegressor(objective='reg:linear',
        n_estimators=10, seed=123)

In [9]: xg_reg.fit(X_train, y_train)

In [10]: preds = xg_reg.predict(X_test)
```

# Trees as base learners example: Scikit-learn API

```
In [11]: rmse = np.sqrt(mean_squared_error(y_test,preds))

In [12]: print("RMSE: %f" % (rmse))
```

```
RMSE: 129043.2314
```

# Linear Base Learners Example: Learning API Only

```
In [1]: import xgboost as xgb
In [2]: import pandas as pd
In [3]: import numpy as np
In [4]: from sklearn.model_selection import train_test_spli

In [5]: boston_data = pd.read_csv("boston_housing.csv")

In [6]: X, y = boston_data.iloc[:,:-1],boston_data.iloc[:,-

In [7]: X_train, X_test, y_train, y_test= train_test_split(
        test_size=0.2, random_state=123)
In [8]: DM_train = xgb.DMatrix(data=X_train,label=y_train)
```

# Linear base learners example: Learning API only

```
In [13]: rmse = np.sqrt(mean_squared_error(y_test,preds))

In [14]: print("RMSE: %f" % (rmse))
```

```
RMSE: 124326.24465
```

# Let's get to work!

EXTREME GRADIENT BOOSTING WITH XGBOOST

# Regularization and base learners in XGBoost

EXTREME GRADIENT BOOSTING WITH XGBOOST

**Sergey Fogelson**
VP of Analytics, Viacom

# Regularization in XGBoost

- Regularization is a control on model complexity
- Want models that are both accurate and as simple as possible
- Regularization parameters in XGBoost:
  - gamma - minimum loss reduction allowed for a split to occur
  - alpha - l1 regularization on leaf weights, larger values mean more regularization
  - lambda - l2 regularization on leaf weights

# L1 Regularization in XGBoost example

```
In [1]: import xgboost as xgb
In [2]: import pandas as pd
In [3]: boston_data = pd.read_csv("boston_data.csv")
In [4]: X,y = boston_data.iloc[:,:-1],boston_data.iloc[:,-1]
In [5]: boston_dmatrix = xgb.DMatrix(data=X,label=y)
In [6]: params={"objective":"reg:linear","max_depth":4}
In [7]: l1_params = [1,10,100]
In [8]: rmses_l1=[]
In [9]: for reg in l1_params:
   ...:     params["alpha"] = reg
   ...:     cv_results = xgb.cv(dtrain=boston_dmatrix,
   ...:     params=params,nfold=4,
   ...:     num_boost_round=10,metrics="rmse",as_pandas=True,seed=123)
   ...:     rmses_l1.append(cv_results["test-rmse-mean"] \
            .tail(1).values[0])
In [10]: print("Best rmse as a function of l1:")
In [11]: print(pd.DataFrame(list(zip(l1_params,rmses_l1)),
         columns=["l1","rmse"]))
```

```
Best rmse as a function of l1:
        l1          rmse
    0    1  69572.517742
    1   10  73721.967141
    2  100  82312.312413
```

# Base Learners in XGBoost

- Linear Base Learner:
    - Sum of linear terms
    - Boosted model is weighted sum of linear models (thus is itself linear)
    - Rarely used
- Tree Base Learner:
    - Decision tree
    - Boosted model is weighted sum of decision trees (nonlinear)
    - Almost exclusively used in XGBoost

# Creating DataFrames from multiple equal-length lists

- ```
  pd.DataFrame(list(zip(list1,list2)),columns=
  ["list1","list2"]))
  ```
- `zip` creates a `generator` of parallel values:
  - `zip([1,2,3],["a","b""c"])` = `[1,"a"],[2,"b"],[3,"c"]`
  - `generators` need to be completely instantiated before they can be used in `DataFrame` objects
- `list()` instantiates the full generator and passing that into the `DataFrame` converts the whole expression
  ```

# Let's practice!

EXTREME GRADIENT BOOSTING WITH XGBOOST