Numeric features

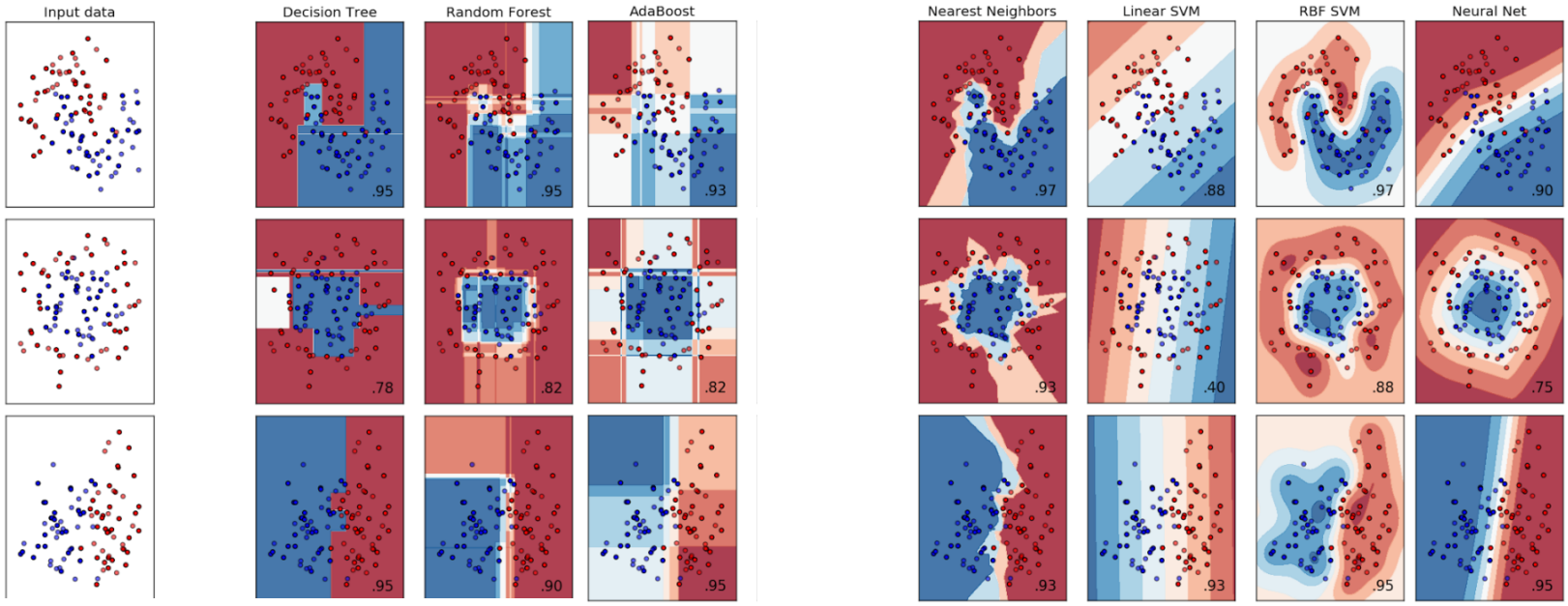
Numeric

- Preprocessing
  a) Tree-based models
  b) Non-tree-based models
- Feature generation

Preprocessing

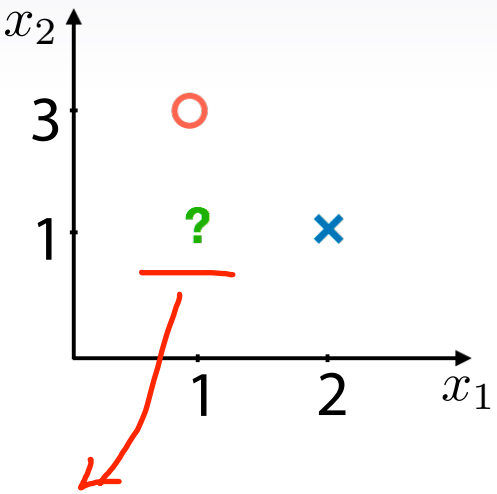
Tree-based models

Non-tree-based models

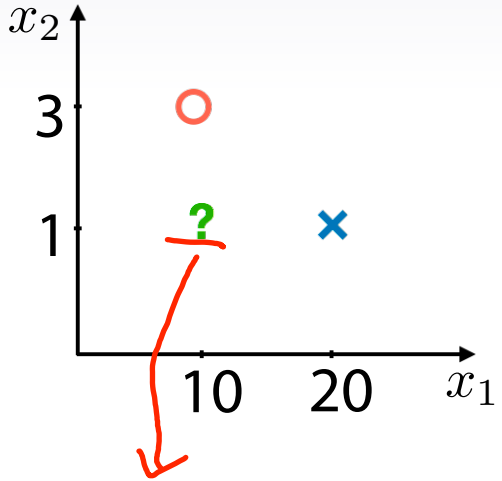


Classifier comparison¶, http://scikit - learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Preprocessing: scaling



closer to X

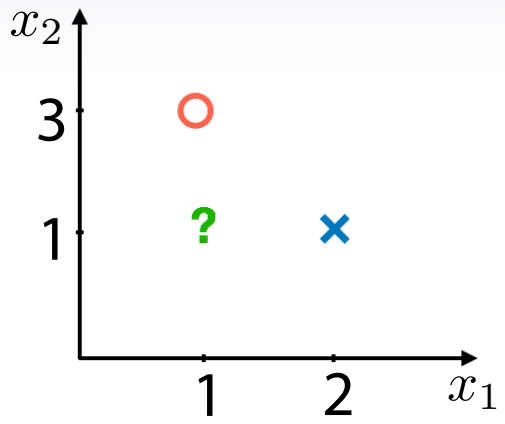


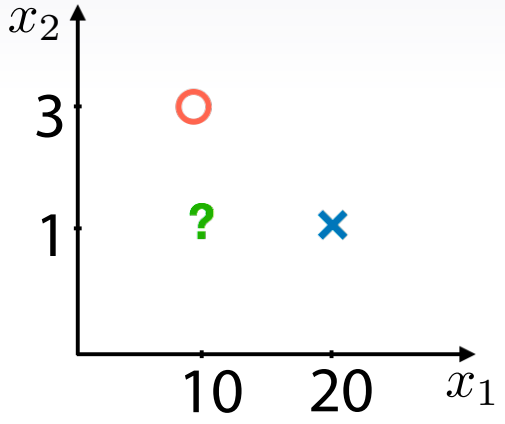
closer to red O

Linear models are also experiencing difficulties with differently scaled features. First, we want regularization to be applied to linear models coefficients for features in equal amount. But in fact, regularization impact turns out to be proportional to feature scale. And second, gradient descent methods can go crazy without a proper scaling.

It is important to understand that different features scalings result in different models quality. In this sense, it is just another hyper parameter you need to optimize. The easiest way to do this is to rescale all features to the same scale.

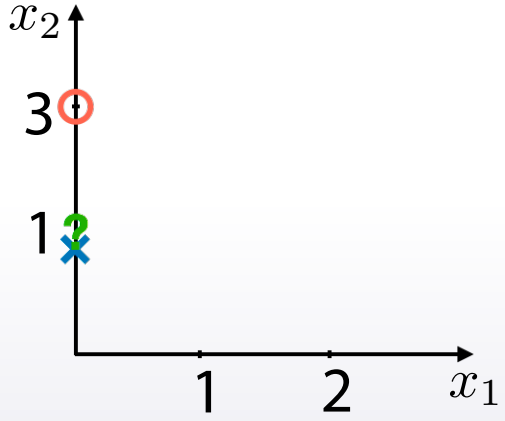
Preprocessing: scaling

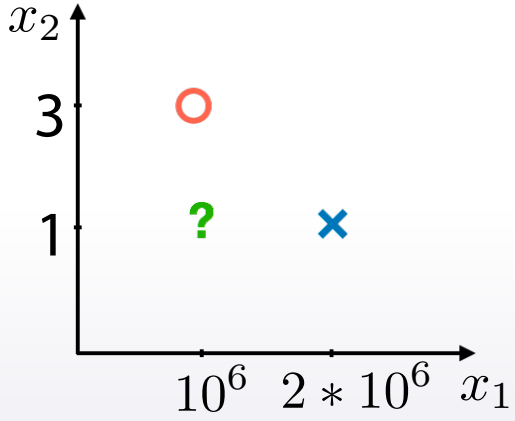




$$x_1 = x_1 * 0$$

$$x_1 = x_1 * 10^6$$





Preprocessing: scaling

1. To [0,1]

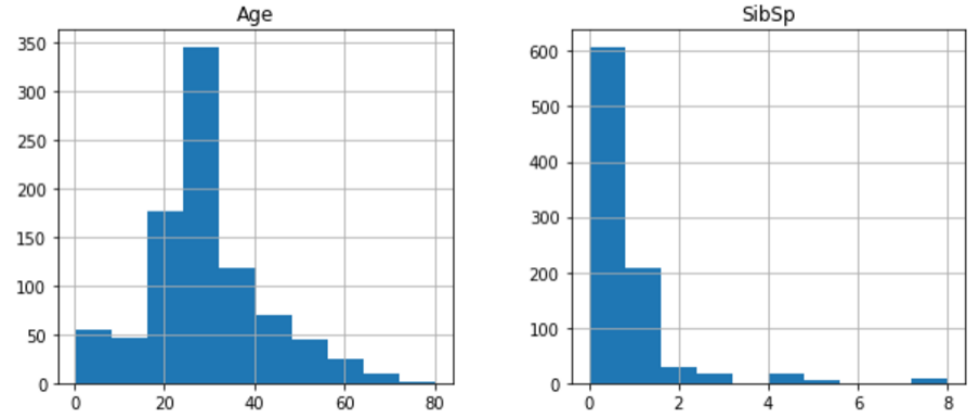
   `sklearn.preprocessing.MinMaxScaler`

$$X = (X - X.min())/(X.max() - X.min())$$

Preprocessing: scaling

$$X = (X - X.min())/(X.max() - X.min())$$

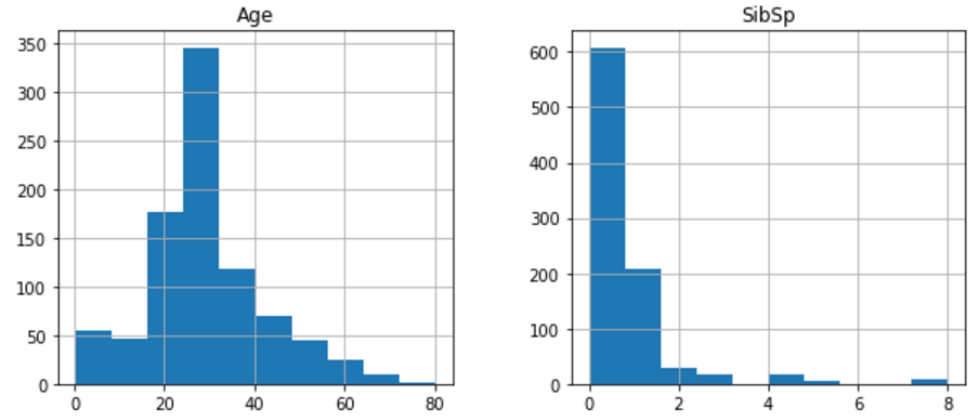
```
train[['Age', 'SibSp']].hist(figsize=(10,4));
```



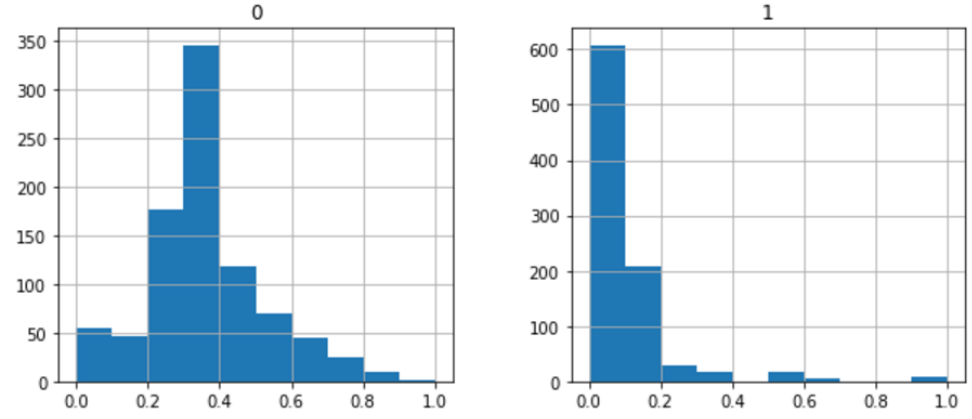
Preprocessing: scaling

$$X = (X - X.min())/(X.max() - X.min())$$

```
train[['Age', 'SibSp']].hist(figsize=(10,4));
```




```
scaler = MinMaxScaler()
xtrain = scaler.fit_transform(train[['Age', 'SibSp']])
pd.DataFrame(xtrain).hist(figsize=(10,4));
```



Preprocessing: scaling

1. To [0,1]

`sklearn.preprocessing.MinMaxScaler`

$$X = (X - X.min())/(X.max() - X.min())$$

2. To mean=0, std=1

`sklearn.preprocessing.StandardScaler`

$$X = (X - X.mean())/X.std()$$

After either of MinMaxScaling or StandardScaling transformations, features impacts on non-tree-based models will be roughly similar. Even more, if you want to use KNN, we can go one step ahead and recall that the bigger feature is, the more important it will be for KNN.

Numeric features

In general case, should we apply these transformations to all numeric features, when we train a non-tree-based model?

- (•) Yes, we should apply a chosen transformation to all numeric features

Correct

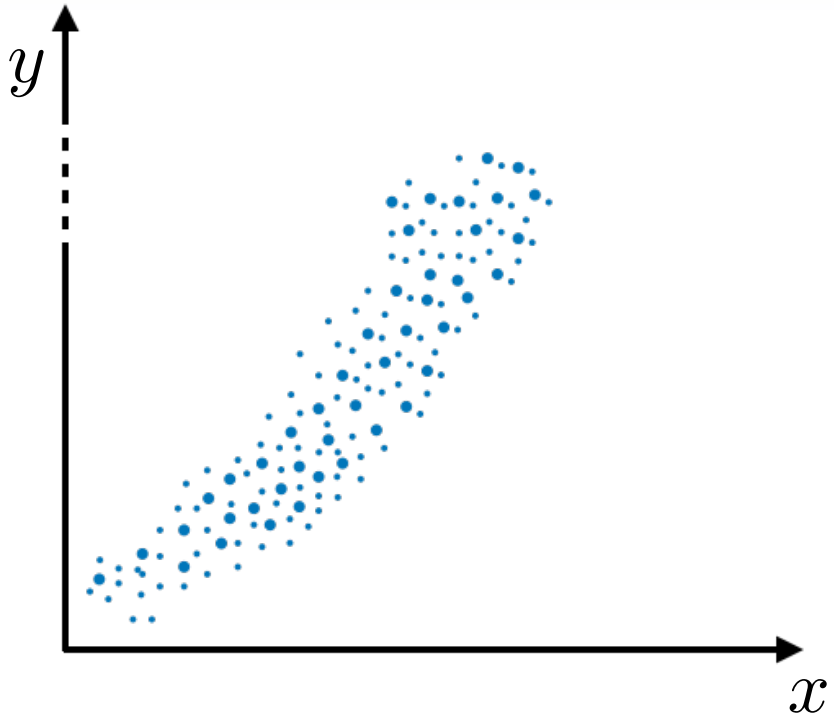
Correct! We use preprocessing to scale all features to one scale, so that their initial impact on the model will be roughly similar. For example, as in the recent example where we used KNN for prediction, this could lead to the case where some features will have critical influence on predictions.

- ( ) No, we should apply it only to few numeric features, leaving other numeric features as they were

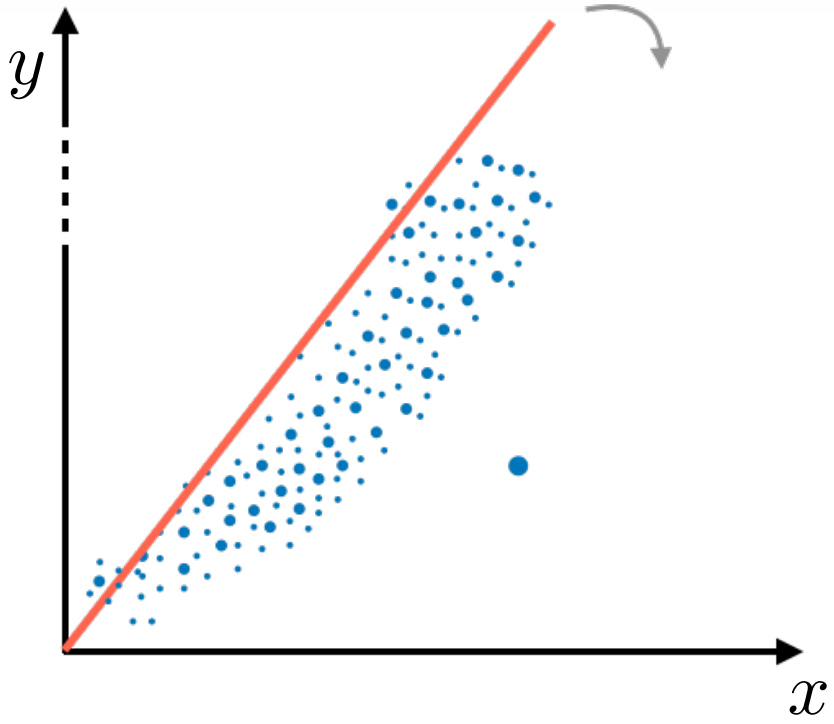
Continue

Preprocessing: outliers

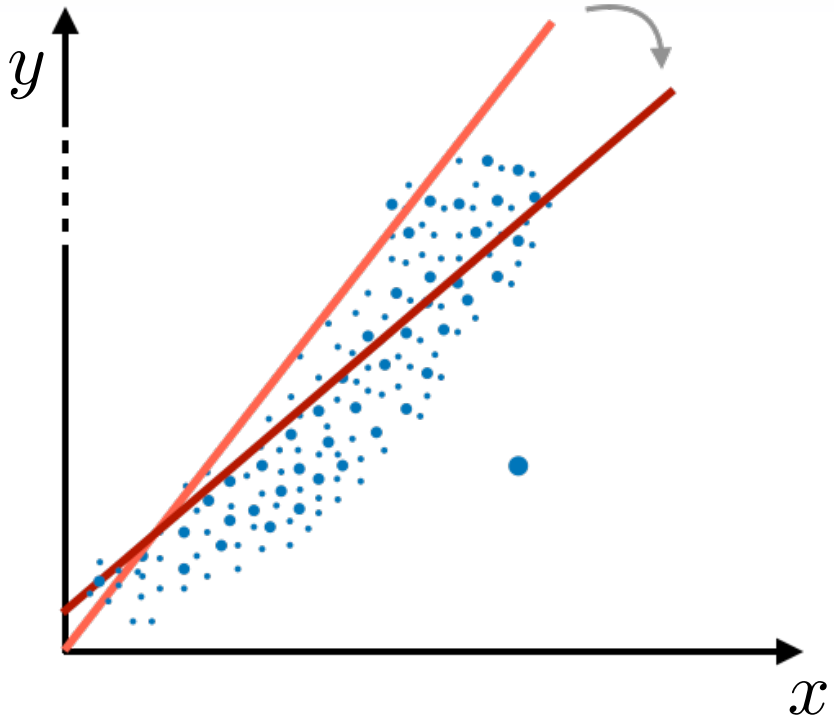
When we work with linear models, there is another important moment that influences model training results. I'm talking about outiers.

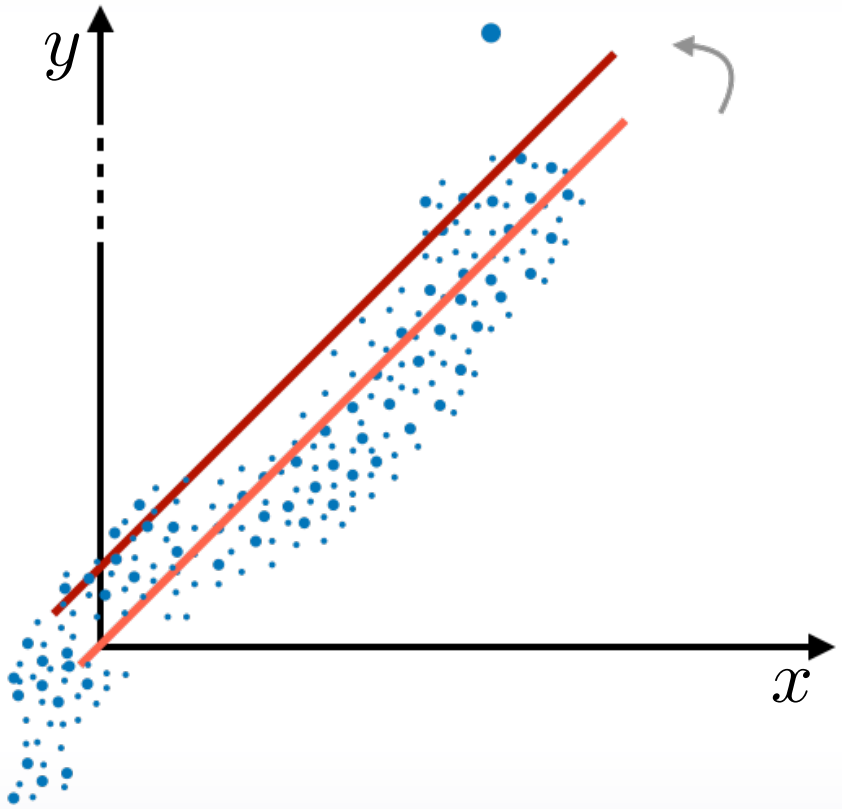


Preprocessing: outliers

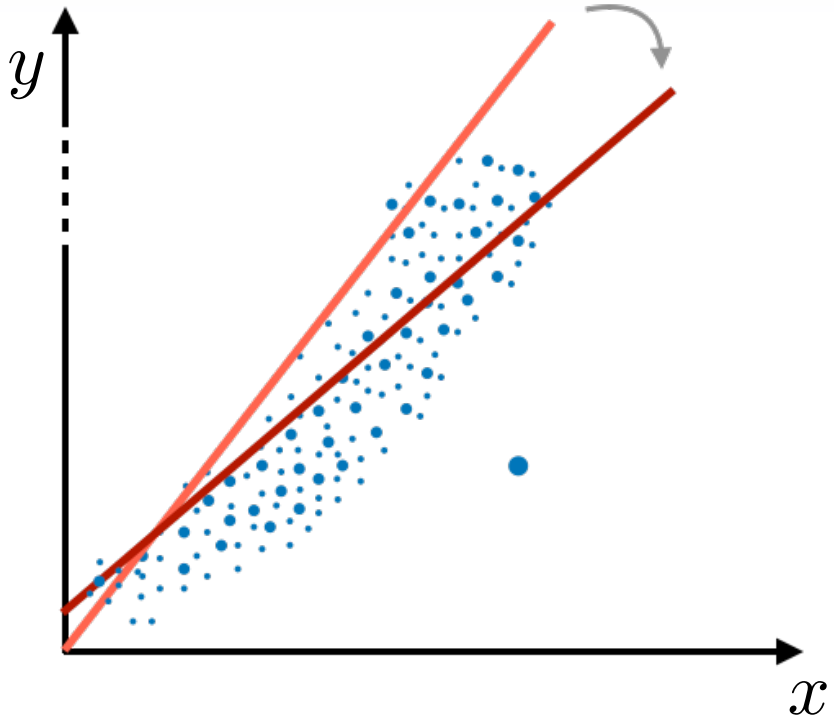


Preprocessing: outliers

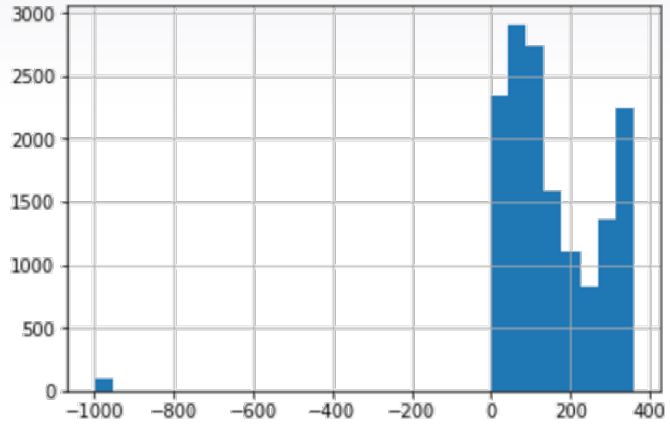




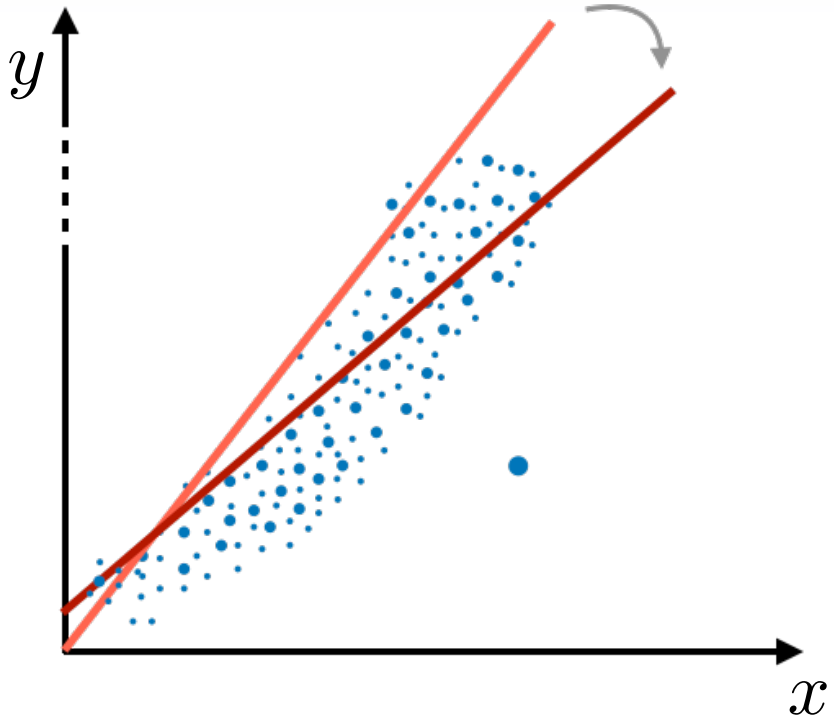
Preprocessing: outliers




```
In [17]: pd.Series(x).hist(bins=30) ;
```

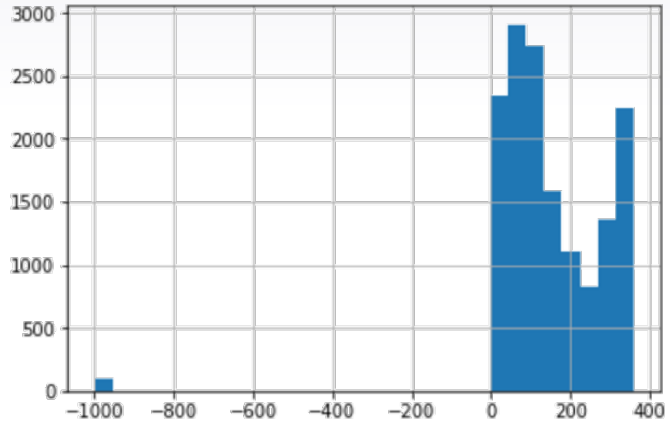


Preprocessing: outliers

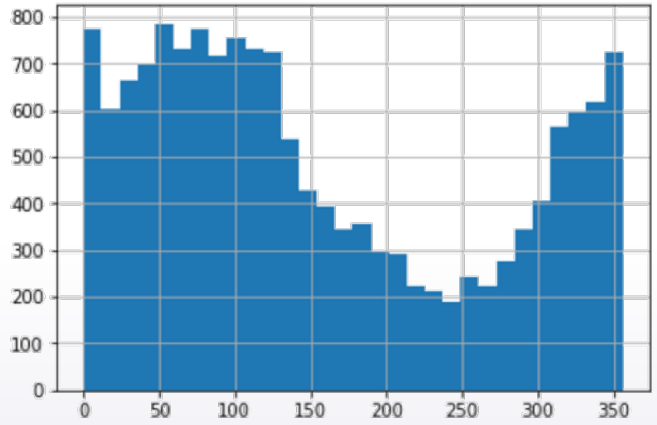


To protect linear models from outliers, we can clip features values between two chosen values of lower bound and upper bound. We can choose them as some percentiles of that feature. For example, first and 99s percentiles. This procedure of clipping is well-known in financial data and it is called winsorization.

```
In [17]: pd.Series(x).hist(bins=30) ;
```




```
In [18]: UPPERBOUND, LOWERBOUND = np.percentile(x, [1, 99])
y = np.clip(x, UPPERBOUND, LOWERBOUND)
pd.Series(y).hist(bins=30) ;
```



Preprocessing: rank

Another effective preprocessing for numeric features is the rank transformation. Basically, it sets spaces between proper assorted values to be equal. This transformation, for example, can be a better option than MinMaxScaler if we have outliers, because rank transformation will move the outliers closer to other objects.

Preprocessing: rank

- `rank([-100, 0, 1e5]) == [0,1,2]`
- `rank([1000,1,10]) = [2,0,1]`

Preprocessing: rank

- `rank([-100, 0, 1e5]) == [0,1,2]`
- `rank([1000,1,10]) = [2,0,1]`

`scipy.stats.rankdata`

Rank can be imported as a random data function from scipy. One more important note about the rank transformation is that to apply to the test data, you need to store the creative mapping from features values to their rank values. Or alternatively, you can concatenate, train, and test data before applying the rank transformation.

Preprocessing

There is one more example of numeric features preprocessing which often helps non-tree-based models and especially neural networks. You can apply log transformation through your data, or there's another possibility. You can extract a square root of the data. Both these transformations can be useful because they drive too big values closer to the features' average value. Along with this, the values near zero are becoming a bit more distinguishable. Despite the simplicity, one of these transformations can improve your neural network's results significantly.

1. Log transform:
   np.log(1 + x)

2. Raising to the power < 1:
   np.sqrt(x + 2/3)

it is beneficial to train a model on concatenated data frames produced by different preprocessings, or to mix models training differently-preprocessed data. Again, linear models, KNN, and neural networks can benefit hugely from this. To this end, we have discussed numeric feature preprocessing, how model choice impacts feature preprocessing, and what are the most commonly used preprocessing methods.

Feature generation

Ways to proceed:

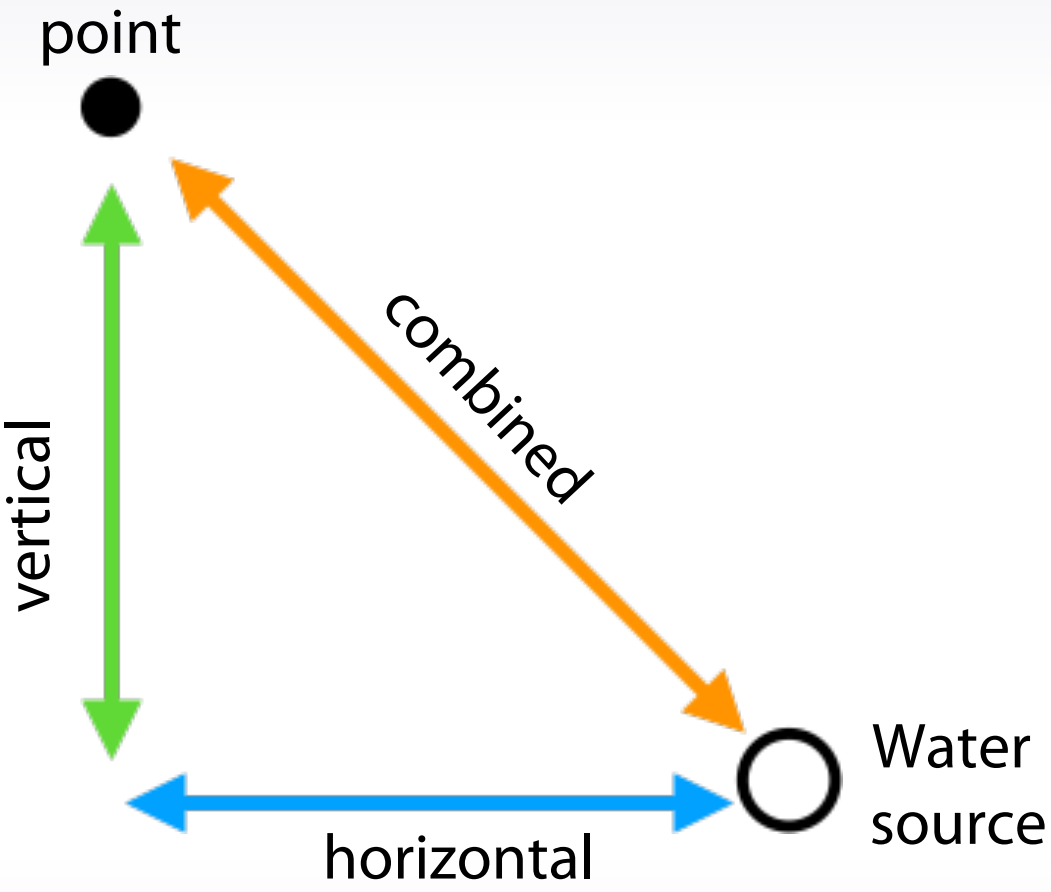
- prior knowledge
- EDA

Feature generation is a process of creating new features using knowledge about the features and the task. It helps us by making model training more simple and effective. Sometimes, we can engineer these features using prior knowledge and logic. Sometimes we have to dig into the data, create and check hypothesis, and use this derived knowledge and our intuition to derive new features.

Feature generation

Squared area: 55 m^2
Price: 107000 $
Price for 1m2: 107000 $ / 55 m2

Feature generation



Combined = (horizontal ** 2 + vectical ** 2) ** 0.5

Feature generation

price	fractional_part
0.99	0.99
2.49	0.49
1.0	0.0
9.99	0.99

Sometimes, if we have prices of products as a feature, we can add new feature indicating fractional part of these prices. For example, if some product costs 2.49, the fractional part of its price is 0.49. This feature can help the model utilize the differences in people's perception of these prices.

Conclusion

1. Scaling and Rank for numeric features:
    a. Tree-based models doesn't depend on them
    b. Non-tree-based models hugely depend on them
2. Most often used preprocessings are:
    a. MinMaxScaler - to [0,1]
    b. StandardScaler - to mean==0, std==1
    c. Rank - sets spaces between sorted values to be equal
    d. np.log(1+x) and np.sqrt(1+x)
3. Feature generation is powered by:
    a. Prior knowledge
    b. Exploratory data analysis