Supervised Learning

Classification

- In binary classification we often speak of one class being the positive class and the other class being the negative class.
- Here, positive doesn't represent having benefit or value, but rather what the object of the study is. So, when looking for spam, "positive" could mean the spam class.
- Which of the two classes is called positive is often a subjective matter, and specific to the domain.

Supervised Learning

- Types of Supervised Machine Learning Algorithms
  - Classification
  - Regression
- **Classification**
  - The goal is to predict a class label from a predefined list of possibilities.
  - **Binary Classification**
    - distinguishing between exactly two classes
  - **Multi-Class Classification**
    - distinguishing between more than two classes
  - **Multi Label Classification**
    - Multi Label classification is also known as multi-output classification are variants of the classification problem where multiple labels may be assigned to each instance.

Regression

- The goal is to predict a **continuous number**, or a **floating point number** in programming terms (a real number in mathematical terms).
    - Ex: Predicting a person's annual income from their education, their age and where they live, is an example of a regression task.

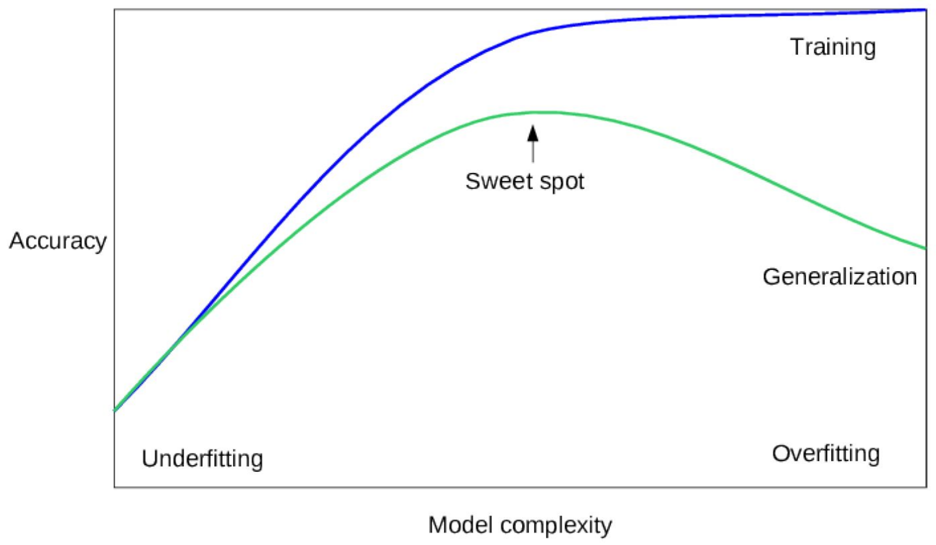
How to determine whether a given is a Classification or Regression task?

- An easy way to distinguish between classification and regression tasks is to ask whether there is some kind of ordering or continuity in the output.
- If there is an ordering, or a continuity between possible outcomes, then the problem is a regression problem.
  - Ex: There is a clear ordering of “making more money” or “making less money”. There is a natural understanding that $40,000 per year is between $50,000 per year and $30,000 per year. There is also a continuity in the output. Whether a person makes 40,000$ or 40,001$

Generalization, Overfitting and Underfitting

- In supervised learning, we build a model on the training data, and then we try to make accurate predictions on new, unseen data, that has the same characteristics as the training set that we used.
- **Overfitting**: Building a complex model that does well on the training set but does not generalize to new data is known as overfitting, because we are **focusing too much on the particularities of the training data**.
- **Underfitting**: Using too simple model is called underfitting, because we don't explain the target output for the training data well enough.

Model Generalization



Model Generalization

- If we choose use a **model that is too simple**, we will do **badly on the training set**, and similarly **badly on the test set**, as we would using only the mean prediction.
- The more complex we allow our model to be, the better we will be able to predict on the training data. However, if our **model becomes too complex**, we start **focusing too much on the particularities of our training set**, and the **model will not generalize well to new data.**
- There is a sweet spot in between, which will yield the best generalization performance. That is the model we want to find.

Supervised Machine Learning Algorithms

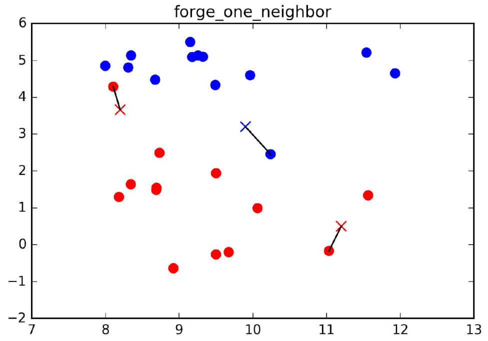
- What are we going to cover?
  - Discuss strength and weaknesses of each algorithm
  - What kind of data they can be best applied to
  - Explain the meaning of the most important parameters and options.
  - Describe the classification and a regression variant of algorithms (if available).
  - Use several datasets to illustrate the different algorithms. Some of the datasets will be small synthetic (meaning made-up) datasets, designed to highlight particular aspects of the algorithms. Other datasets will be larger, real world examples datasets.
    - Example: An example of a synthetic two-class classification dataset is the forge dataset, which has two features.

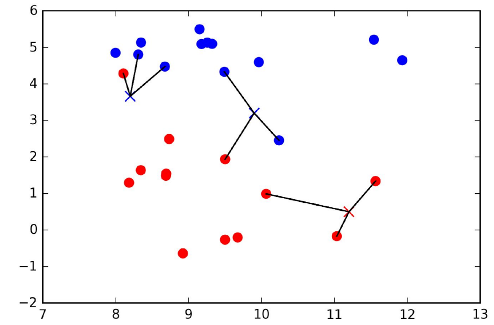
k-Nearest Neighbor

- The k-Nearest Neighbors (kNN) algorithm is arguably the simplest machine learning algorithm.
- Building the model only consists of storing the training dataset.
- To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset, it “nearest neighbors”.

k-Neighbors Classification

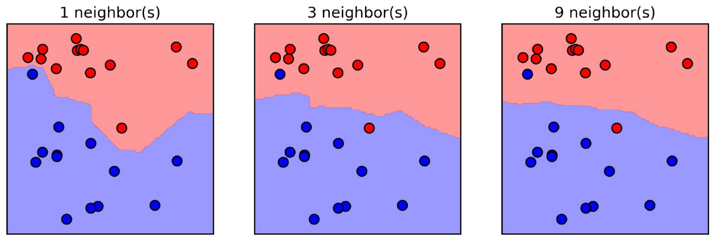
- In its simplest version, the algorithm only considers exactly one nearest neighbor, which is the **closest training data point to the point we want to make a prediction for**.
- The prediction is then simply the known output for this training point.
- When considering more than one neighbor, we use **voting to assign a label**. This means, for each test point, we count how many neighbors are red, and how many neighbors are blue. We then assign the class that is more frequent: in other words, the majority class among the k neighbors.





Analyzing KNeighborsClassifier

- As you can see in the figure, using a single neighbor results in a decision boundary that follows the training data closely.
- Considering more and more neighbors leads to a smoother decision boundary.
- A smoother boundary corresponds to a simple model.
- In other words, using **few neighbors corresponds to high model complexity** (as shown in the right side figure) and using **many neighbors corresponds to low model complexity.**



k-Neighbors Regression

- The prediction using a single neighbor is just the target value of the nearest neighbor.
- When using **multiple nearest neighbors for regression, the prediction is the average (or mean) of the relevant neighbors.**
- We can evaluate the regression model using the score method, which for regressors returns the R^2 score.
- The R^2 score, also known as **coefficient of determination, is a measure of goodness of a prediction for a regression model, and yields a score up to 1.**
- A value of **1 corresponds to a perfect prediction**, and **a value of 0 corresponds to a constant model that just predicts the mean of the training set** responses.

Parameters of k-Nearest Neighbors

- In principal, there are two important parameters to the K-Neighbors classifier:
    - **The number of neighbors**
    - **How you measure distance between data points.**
- In practice, **using a small number of neighbors** like 3 or 5 often **works well**, but you should **certainly adjust this parameter**.
- **By default, Euclidean distance is used**, which works well in many settings.

Strengths of k-Nearest Neighbors

- It is very easy to understand
- Often **gives reasonable performance without a lot of adjustments.**
- Using nearest neighbors is a **good baseline method** to try before considering more advanced techniques.
- **Building the nearest neighbors model is usually very fast, but when your training set is very large (either in number of features or in number of samples) prediction can be slow.**

Weaknesses of k-Nearest Neighbors

- When using nearest neighbors, it’s **important to preprocess your data.**
- Nearest neighbors often **does not perform well on dataset with very many features**, in particular **sparse datasets**, a common type of data in which **there are many features, but only few of the features are non-zero for any given data point.**
- So while the nearest neighbors algorithm is easy to understand, **it is not often used in practice, due to prediction being slow, and its inability to handle many features.**

Note : The method we discuss next has neither of these drawbacks.

Linear Models

Linear Models

- Linear models are a class of models that are widely used in practice, and have been studied extensively in the last few decades, with roots going back over a hundred years.
- Linear models are models that make a **prediction that using a** **linear function of the input features.**
- Linear models for regression can be characterized as regression models for which the prediction is a line for a **single feature,** a plane when using two features, or a hyperplane in higher dimensions (that is when **having more features**).

Linear Models for Regression

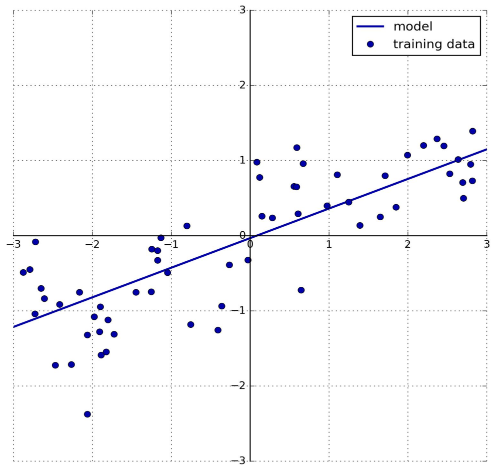
- For regression, the general prediction formula for a linear model looks as follows:

  ŷ = w[0] * x[0] + w[1] * x[1] + … + w[p] * x[p] + b

  of a single data point, w and b are parameters of the model that are learned, and ŷ is the prediction the model makes.
- For a dataset with a single feature, this is:

  ŷ = w[0] * x[0] + b

  Where w[0] is the slope and b is the y-axis offset.



Linear Models for Regression

- For datasets with many features, linear models can be very powerful. In particular, if you have more features than training data points, any target y can be perfectly modeled (on the training set) as a linear function.
- There are many different linear models for regression. The difference between these models lies in how the model parameters w and b are learned from the training data, and how model complexity can be controlled.

Linear regression (aka Ordinary Least Squares)

$$Cost\ (W) = RSS(W) = \sum_{i=1}^{N} \{y_i - \hat{y}_i\}^2 = \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2$$

- Linear regression, or ordinary least squares (OLS), is the simplest and most classic linear method for regression.
- Linear regression finds the parameters w and b that minimize the mean squared error between predictions and the true regression targets, y, on the training set.
- Linear regression has no parameters, which is a benefit, but it also has no way to control model complexity.
- The "slope" parameters (w), also called weights or coefficients and (b) is offset or intercept.
- For datasets with many features, linear models can be very powerful. In particular, if we have more features than training data points, any target y can be perfectly modeled (on the training set) as a linear function.
- One of the **most commonly used alternatives to standard linear regression is ridge regression.**

Linear Regression - Demo

Ridge Regression

$$Cost\,(W) = RSS(W) + \lambda * (sum\ of\ squares\ of\ weights)$$

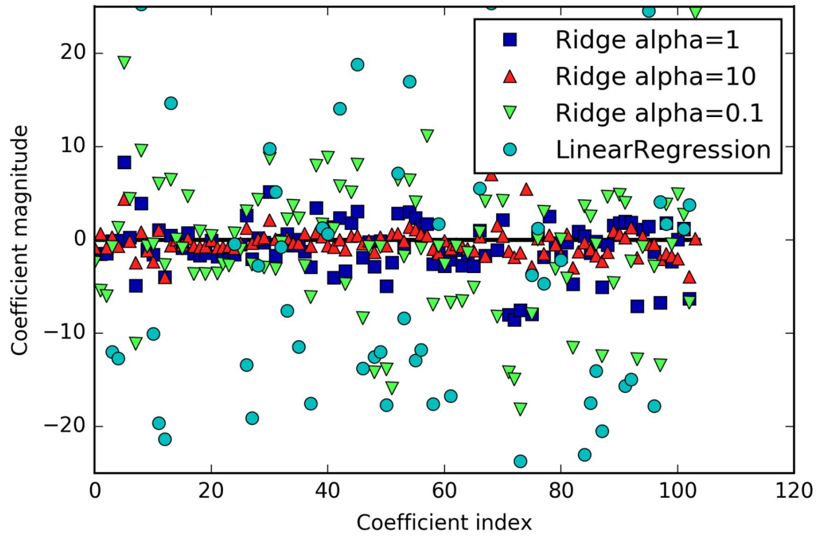
$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

- Ridge regression is also a **linear model for regression**, so the formula it uses to make predictions is the same one used for ordinary least squares.
- In ridge regression, though, **the coefficients (w) are chosen not only so that they predict well on the training data, but also to fit an additional constraint.** We also want the **magnitude of coefficients to be as small as possible**; in other words, **all entries of w should be close to zero.**
- Intuitively, this means **each feature should have as little effect on the outcome as possible** (which translates to having a small slope), while still predicting well.
- This constraint is an example of what is called **regularization**.
- Regularization means **explicitly restricting a model to avoid overfitting**.
- **Ridge regression uses L2 Regularization to avoid overfitting.**

Ridge Regression

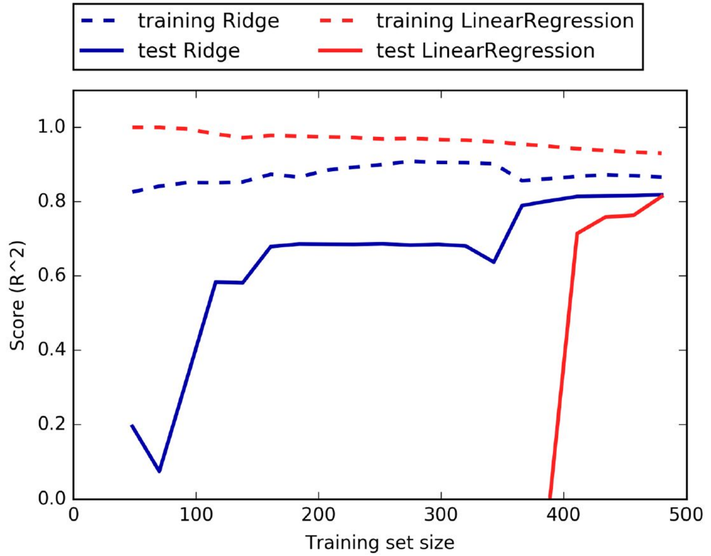
- Ridge is a more restricted model, so we are less likely to overfit.
- A less complex model means worse performance on the training set, but better generalization.
- The Ridge Regression model makes a trade-off between the simplicity of the model (near-zero coefficients) and its performance on the training set.
- How much importance the model places on simplicity versus training set performance can be specified by the user, using the alpha parameter.
- The optimum setting of alpha depends on the particular dataset we are using.
- Increasing alpha forces coefficients to move more toward zero, which decreases training set performance but might help generalization.

Impact of Regularization on Ridge Regression



- A **higher alpha means a more restricted model**, so we expect the entries of coefficients to have smaller magnitude for a high value of alpha than for a low value of alpha.

Impact of Regularization on Ridge Regress



Learning curves for ridge regression and linear regression on the Boston Housing dataset
Scenario: Alpha is fixed and varied the amount of training data

Key TakeAways

- The training score is higher than the test score for all dataset sizes, for both ridge and linear regression.
- As ridge is regularized, the training score of ridge is lower than the training score for linear regression across the board.
- The test score for ridge is better, particularly for small subsets of the data. For less than 400 data points, linear regression is not able to learn anything. As more and more data becomes available to the model, both models improve, and linear regression catches up with ridge in the end.
- The lesson here is that with enough training data, regularization becomes less important, and given enough data, ridge and linear regression will have the same performance
- From previous fig we could see that decrease in training performance for linear regression. If more data is added, it becomes harder for a model to overfit, or memorize the data.

Ridge Regression - Demo

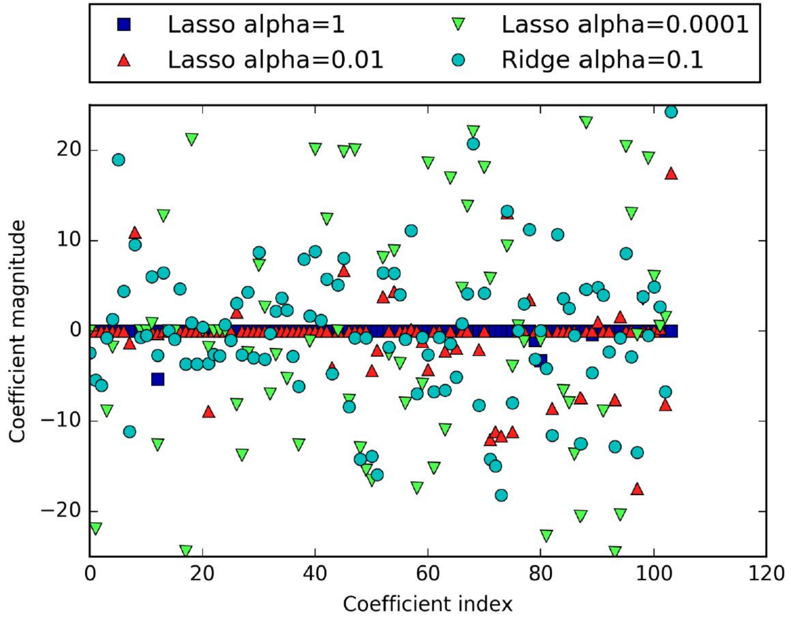
Lasso

$$Cost\ (W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} |w_j|$$

- Similar to ridge regression, lasso also restricts coefficients to be close to zero, but in a slightly different way, called L1 regularization.
- The consequence of L1 regularization is that when using the lasso, some coefficients are exactly zero. This means some features are entirely ignored by the model. This can be seen as a form of automatic feature selection.
- Having some coefficients be exactly zero often makes a model easier to interpret, and can reveal the most important features of your model.
- Similarly to Ridge, the Lasso also has a regularization parameter, alpha, that controls how strongly coefficients are pushed toward zero.
- The lasso penalizes the L1 norm of the coefficient vector—or in other words, the sum of the absolute values of the coefficients.
- If we set alpha too low, however, we again remove the effect of regularization and end up overfitting, with a result similar to LinearRegression.

Impact of Regularization on Lasso Regression



When to use what???

- In practice, ridge regression is usually the first choice between these two models (Ridge and Lasso)
- However, if we have a large amount of features and expect only a few of them to be important, Lasso might be a better choice.
- Similarly, if we would like to have a model that is easy to interpret, Lasso will provide a model that is easier to understand, as it will select only a subset of the input features.
- *Note: scikit-learn provides the ElasticNet class, which combines the penalties of Lasso and Ridge. In practice, this combination works best, though at the price of having two parameters to adjust: one for the L1 regularization, and one for the L2 regularization.*

Lasso Regression - Demo

Linear Models for Classification

Linear Models for Classification

- Linear models are also extensively used for classification.
- Let's look at binary classification first. In this case, a prediction is made using the following formula: $\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b > 0$
- The above formula looks very similar to the one for linear regression, but instead of just returning the weighted sum of the features, we threshold the predicted value at zero.
- If the function is smaller than zero, we predict the class –1; if it is larger than zero, we predict the class +1.
- This **prediction rule is common to all linear models for classification**. Again, there are many different ways to find the coefficients (w) and the intercept (b).

Linear Models for Classification

- For linear models for regression, the output, ŷ, is a linear function of the features: a line, plane, or hyperplane (in higher dimensions).
- For linear models for classification, the decision boundary is a linear function of the input. In other words, a (binary) linear classifier is a classifier that separates two classes using a line, a plane, or a hyperplane.
- There are many algorithms for learning linear models. These algorithms all differ in the following two ways:
    - The way in which they measure how well a particular combination of coefficients and intercept fits the training data
    - If and what kind of regularization they use

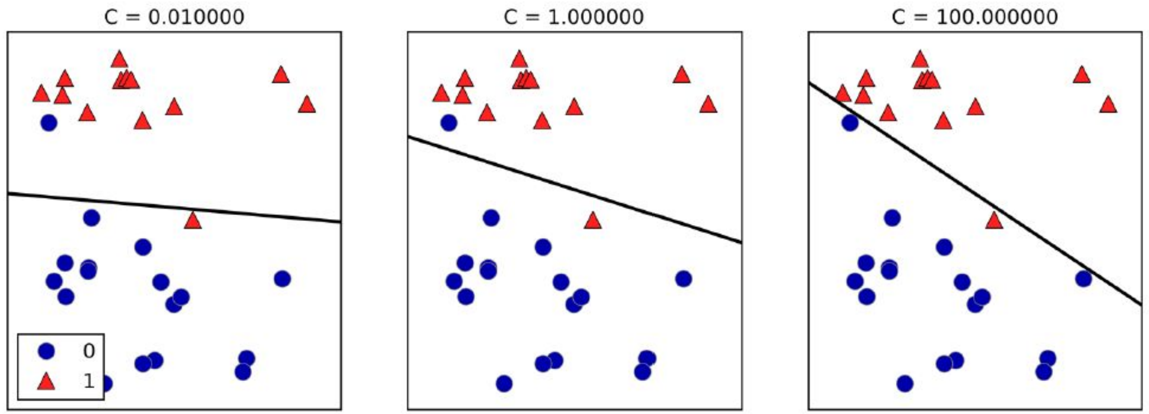
Linear Models for Classification

- Different algorithms choose different ways to measure what fitting the training set well means.
- The two most common linear classification algorithms are Logistic Regression, and Linear Support Vector Machines
- Despite its name, LogisticRegression is a classification algorithm and not a regression algorithm, and it should not be confused with LinearRegression.

Linear Models for Classification

- By default, both models Logistic Regression and Linear Support Vector Classifier apply an L2 regularization, in the same way that Ridge does for regression.
- For LogisticRegression and LinearSVC the trade-off parameter that determines the strength of the regularization is called C, and higher values of C correspond to less regularization.
- In other words, when you use a high value for the parameter C, LogisticRegression and LinearSVC try to fit the training set as best as possible, while with low values of the parameter C, the models put more emphasis on finding a coefficient vector (w) that is close to zero.
- The low values of *C* will cause the algorithms to try to adjust to the majority of data points, while using a higher value of *C* stresses the importance that each individual data point be classified correctly.

Decision boundaries of a linear SVM for different values of C



- A very small C corresponding to a lot of regularization.
- For large values of C model tries hard to correctly classify all points, but might not capture the overall layout of the classes well. In other words, this model is likely overfitting.

Linear Models for Classification

- Similarly to the case of regression, linear models for classification might seem very restrictive in low-dimensional spaces, only allowing for decision boundaries that are straight lines or planes.
- Again, in high dimensions, linear models for classification become very powerful, and guarding against overfitting becomes increasingly important when considering more features.
- If we desire a more interpretable model, using L1 regularization might help, as it limits the model to using only a few features.
- There are many parallels between linear models for binary classification and linear models for regression.
- As in regression, the main difference between the models is the penalty parameter, which influences the regularization and whether the model will use all available features or select only a subset.

Linear models for multiclass classification

- Many linear classification models are for binary classification only, and don’t extend naturally to the multiclass case (with the exception of logistic regression).
- A common technique to extend a binary classification algorithm to a multiclass classification algorithm is the one-vs.-rest approach.
- In the one-vs.-rest approach, a binary model is learned for each class that tries to separate that class from all of the other classes, resulting in as many binary models as there are classes. To make a prediction, all binary classifiers are run on a test point. The classifier that has the highest score on its single class “wins,” and that class label is returned as the prediction.

Parameters of Linear Models

- The main parameter of linear models is the regularization parameter, called alpha in the regression models and C in LinearSVC and LogisticRegression. Large values for alpha or small values for C mean simple models.
- Usually C and alpha are searched for on a logarithmic scale.
- If we assume that only a few of our features are actually important, you should use L1. Otherwise, you should default to L2.
- *L1 can also be useful if interpretability of the model is important.* As L1 will use only a few features, it is easier to explain which features are important to the model, and what the effects of these features are.

Strengths of Linear Models

- Linear models are very fast to train, and also fast to predict.
- They scale to very large datasets and work well with sparse data.
- If your dataset has highly correlated features; in these cases, the coefficients might be hard to interpret.
- Linear models often perform well when the number of features is large compared to the number of samples. They are also often used on very large datasets, simply because it's not feasible to train other models.
- However, in lower-dimensional spaces, other models might yield better generalization performance.

Naive Bayes Classifier

Agenda

- What’s Naive Bayes Classifier?
- How is it different from Linear Models?
- What’s the math behind Naive Bayes Classifier?
- What does the word ***Naive*** means in Naive Bayes?
- What are the different types of Naive Bayes Classifiers?
- When to use what classifier?
- What are the strengths and weaknesses of Naive Bayes Classifiers?
- Conclusion

Naive Bayes Classifier

- Naive Bayes classifier is a probabilistic machine learning model used for classification task.
- Probabilistic models are tend to be faster in training than linear models.
- The price paid for this efficiency is that Naive Bayes models often provide generalization performance that is slightly worse than that of linear classifiers like LogisticRegression and LinearSVC.
- The reason that naive Bayes models are so efficient is that they learn parameters by looking at each feature individually and collect simple per-class statistics from each feature.
- There are three kinds of naive Bayes classifiers implemented in scikit-learn:
  - GaussianNB
  - BernoulliNB
  - MultinomialNB

Naive Bayes Classifier

- Bayes Theorem is the crux of Naive Bayes Classifier.
- Using Bayes theorem we can find the probability of A happening, given that B occured.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Here A is the Hypothesis and B is the Evidence.
- The assumption made in here is that features or the predictors are independent of each other, i.e., the presence or absence of one feature doesn't impact the other. Another assumption made here is that all the features or predictors will have the equal effect on the outcome. Hence it's called Naive.

Naive Bayes - Example

	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY GOLF
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes

Math behind Naive Bayes Classifier

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \qquad X = (x_1, x_2, x_3, ..., x_n)$$

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

- For all entries in the dataset, the denominator does not change, it remain static. Therefore, the denominator can be removed and a proportionality can be introduced.

$$P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

- The class variable (y) has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class y with maximum probability.

Types of Naive Bayes Classifiers

- Multinomial Naive Bayes
  - Assumes count data (that is, that each feature represents an integer count of something, like how often a word appears in a sentence).
  - Mostly used for Document Classification.
- Bernoulli Naive Bayes
  - Assumes that the features are boolean variables.
  - The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.
- Gaussian Naive Bayes
  - Applied to any continuous data
  - Stores the average value as well as the standard deviation of each feature for each class.
- Complement Naive Bayes
  - CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets.
  - CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks.

Strengths and Weaknesses

- Gaussian Naive Bayes is mostly used on very high-dimensional data, while Multinomial Naive Bayes and Bernoulli Naive Bayes are widely used for sparse count data such as text.
- Multinomial Naive Bayes usually performs better than Bernoulli Naive Bayes, particularly on datasets with a relatively large number of nonzero features (i.e., large documents).
- The naive Bayes models share many of the strengths and weaknesses of the linear models. They are very fast to train and to predict, and the training procedure is easy to understand.
- Naive Bayes models are great baseline models and are often used on very large datasets, where training even a linear model might take too long.

Naive Bayes Classifiers - Conclusion

- Naive Bayes algorithms are mostly used in
    - Sentiment Analysis
    - Spam Filtering
    - Recommendation Systems etc.
- They are **fast and easy to implement** but their **biggest disadvantage is that the requirement of predictors to be independent.**
- In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

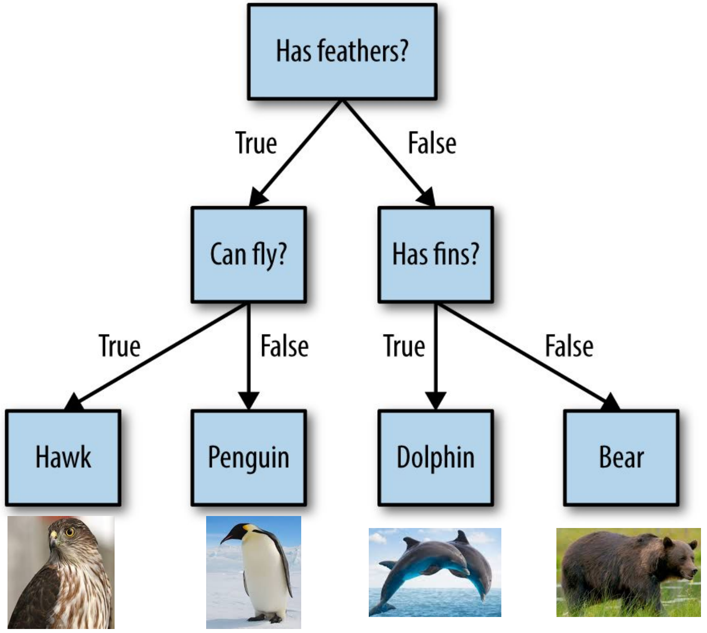
Naive Bayes Classifier - Demo

Decision Trees

Decision Trees

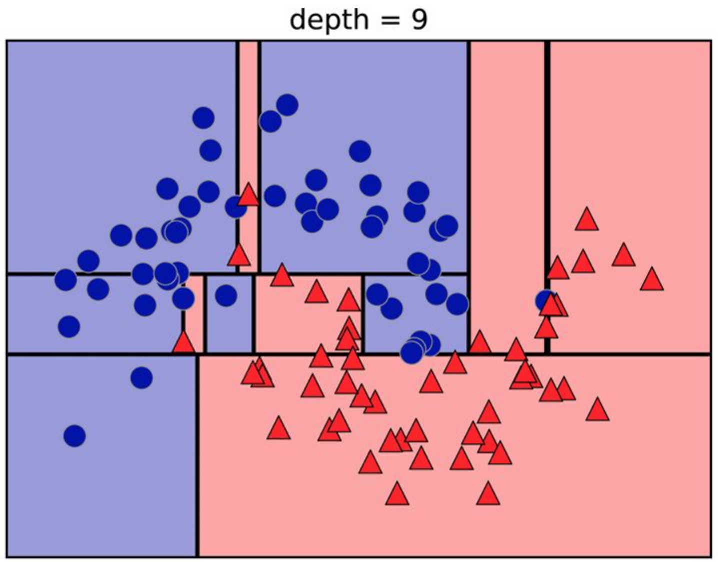
- Decision Trees are widely used models for classification and regression tasks.
- Essentially they learn a hierarchy of if-else questions leading to a decision.
- This series of questions can be expressed as a decision tree.

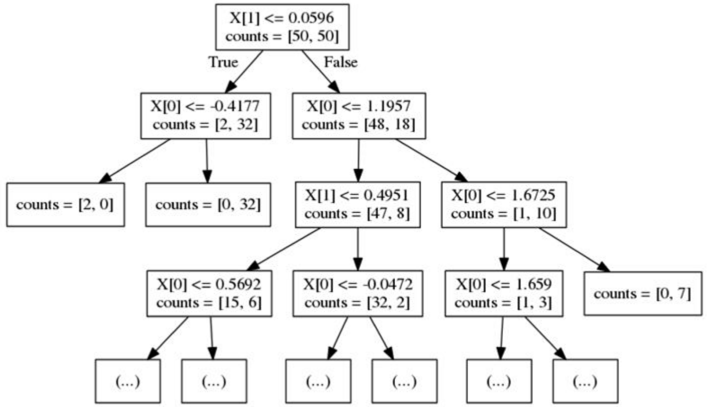
Decision Trees - Example



A decision tree to distinguish among several animals

Complexity of Decision Trees





Decision boundary of tree with depth 9 (left) and part of the corresponding tree (right).
The full tree is quite large and hard to visualize.

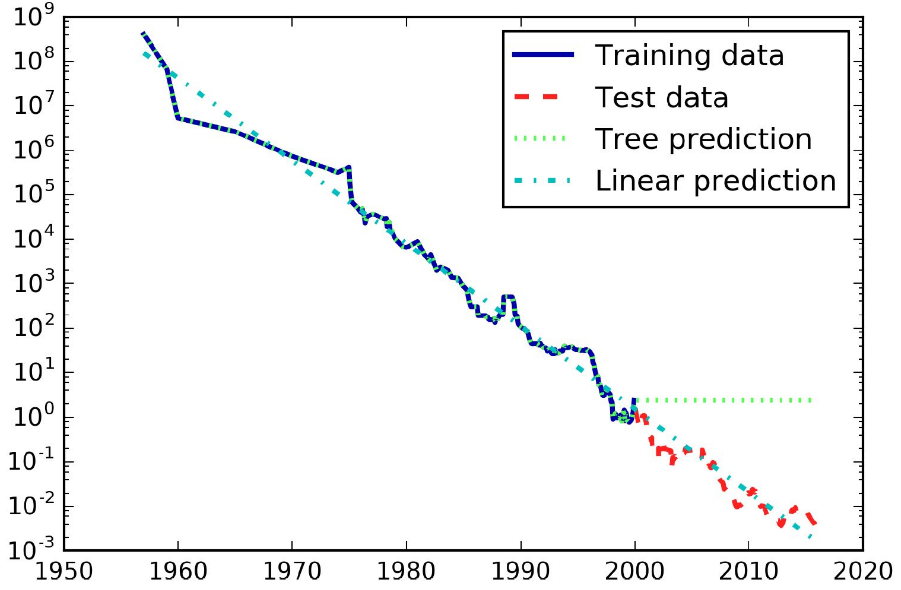
Controlling complexity of Decision Trees

- There are two common strategies to prevent Overfitting:
- Pre-Pruning
  - Stopping the creation of the tree early.
  - Possible criteria for pre-pruning include limiting the maximum depth of the tree,limiting the maximum number of leaves, or requiring a minimum number of points in a node to keep splitting it.
- Post Pruning or Pruning
  - Build the tree but then removing or collapsing nodes that contain little information

Note : scikit-learn only implements pre-pruning, not post-pruning.

Decision Tree Regressor

- The DecisionTreeRegressor (and all other tree-based regression models) is not able to extrapolate, or make predictions outside of the range of the training data.



Comparison of predictions made by a linear model and predictions made by a regression tree on the RAM price data

Strengths and Weaknesses

- Usually, picking one of the pre-pruning strategie setting either max_depth, max_leaf_nodes, or min_samples_leaf—is sufficient to prevent overfitting.
- Decision trees have two advantages over many of the algorithms we've discussed so far:
  - The resulting model can easily be visualized and understood by nonexperts
  - The algorithms are completely invariant to scaling of the data. As each feature is processed separately, and the possible splits of the data don't depend on scaling, no preprocessing like normalization or standardization of features is needed for decision tree algorithms. In particular, decision trees work well when you have features that are on completely different scales, or a mix of binary and continuous features.
- The main downside of decision trees is that even with the use of pre-pruning, they tend to overfit and provide poor generalization performance. Therefore, in most applications, the ensemble methods are usually used in place of a single decision tree.

Decision Trees - Demo

Ensembles of Decision Trees

Ensembles of Decision Trees

- Ensembles are methods that combine multiple machine learning models to create more powerful models.
- There are many models in the machine learning literature that belong to this category, but there are two ensemble models that have proven to be effective on a wide range of datasets for classification and regression, both of which use decision trees as their building blocks:
  - Random Forests
  - Gradient Boosted Decision Trees.

Random Forests

- The main drawback of decision trees is that they tend to overfit the training data. Random forests are one way to address this problem.
- A random forest is essentially a collection of decision trees, where each tree is slightly different from the others.
- The main idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data. If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results.

Random Forests

- Random forests get their name from **injecting randomness into the tree building to ensure each tree is different.**
- There are **two ways in which the trees in a random forest are randomized:**
    - By selecting the **data points used to build a tree**
    - By selecting the **features in each split test**

Building Random Forest

- To build a random forest model, you need to decide on the number of trees to build - number of estimators
- These trees will be built completely independently from each other, and the algorithm will make different random choices for each tree to make sure the trees are distinct.
- To build a tree, we first take what is called a bootstrap sample of our data. That is, from our n_samples data points, we repeatedly draw an example randomly with replacement (meaning the same sample can be picked multiple times), n_samples times.
- Next, a decision tree is built based on this newly created dataset

Building Random Forest

- Instead of looking for the best test for each node, in each node the algorithm randomly selects a subset of the features, and it looks for the best possible test involving one of these features. The number of features that are selected is controlled by the max_features parameter.

Building Random Forest

- A critical parameter in this process is **max_features**.
  - If we set max_features to **n_features**, that means that each split can look at all features in the dataset, and no randomness will be injected in the feature selection (the randomness due to the bootstrapping remains, though).
  - If we set max_features to 1, that means that the splits have no choice at all on which feature to test, and can only search over different thresholds for the feature that was selected randomly.
  - Therefore, a **high max_features means that the trees in the random forest will be quite similar, and they will be able to fit the data easily, using the most distinctive features. A low max_features means that the trees in the random forest will be quite different, and that each tree might need to be very deep in order to fit the data well.**

Building Random Forest

- To make a prediction using the random forest, the algorithm first makes a prediction for every tree in the forest. For regression, we can average these results to get our final prediction.
- For classification, a **soft voting** strategy is used. This means each algorithm makes a “soft” prediction, providing a probability for each possible output label.
- The probabilities predicted by all the trees are averaged, and the class with the highest probability is predicted.

Strengths of Random Forests

- Random forests for regression and classification are currently among the **most widely used machine learning methods.**
- They are very powerful, often work well **without heavy tuning of the parameters, and don't require scaling of the data.**
- Essentially, random forests share all of the benefits of decision trees, while making up for some of their deficiencies.
- One reason to still use decision trees is if you need a compact representation of the decision-making process. It is basically impossible to interpret tens or hundreds of trees in detail, and trees in random forests tend to be deeper than decision trees (because of the use of feature subsets). Therefore, if you need to summarize the prediction making in a visual way to nonexperts, a single decision tree might be a better choice.

Parameters of Random Forest

- We should keep in mind that random forests, by their nature, are random, and setting different random states (or not setting the random_state at all) can drastically change the model that is built.
- The more trees there are in the forest, the more robust it will be against the choice of random state. If you want to have reproducible results, **it is important to fix the random_state.**

Parameters of Random Forest

- The important parameters to adjust are n_estimators, max_features, and possibly pre-pruning options like max_depth.
- For n_estimators
  - Larger is always better.
  - Averaging more trees will yield a more robust ensemble by reducing overfitting.
  - However, there are diminishing returns, and more trees need more memory and more time to train.
  - A common rule of thumb is to build “as many as you have time/memory for.”

Parameters of Random Forest

- The parameter max_features determines how random each tree is
  - A smaller max_features reduces overfitting.
  - In general, it’s a good rule of thumb to use the default values:
    - max_features=sqrt(n_features) for classification
    - max_features=log2(n_features) for regression.
- Adding max_features or max_leaf_nodes might sometimes improve performance. It can also drastically reduce space and time requirements for training and prediction.

Downsides of Random Forests

- Random forests don’t tend to perform well on very high dimensional, sparse data, such as text data. For this kind of data, linear models might be more appropriate.
- Random forests usually work well even on very large datasets, and training can easily be parallelized over many CPU cores within a powerful computer. However, random forests require more memory and are slower to train and to predict than linear models.
- If time and memory are important in an application, it might make sense to use a linear model instead.

Random Forest - Demo

Gradient Boosted Regression Trees a.k.a Gradient Boosting Machines

- Gradient Boosting Machines is an another ensemble method that combines multiple decision trees to create a more powerful model.
- It can be used for regression and classification tasks.
- In contrast to the random forest approach, gradient boosting works by building trees in **a serial manner,** where each tree tries to correct the mistakes of the previous one.
- By default, **there is no randomization in gradient boosted regression trees; instead, strong pre-pruning is used.**
- Gradient boosted trees often use very shallow trees, of depth one to five, which makes the model smaller in terms of memory and makes predictions faster.

Gradient Boosting Machines

- The main idea behind gradient boosting is to combine many simple models (known as weak learners), like shallow trees.
- Each tree can only provide good predictions on part of the data, and so more and more trees are added to iteratively improve performance.
- **Gradient boosted trees are frequently the winning entries in machine learning competitions, and are widely used in industry.**
- They are generally a bit more sensitive to parameter settings than random forests, but can provide better accuracy if the parameters are set correctly.
- Apart from the pre-pruning and the number of trees in the ensemble, another important parameter of gradient boosting is the **learning_rate**, which controls how strongly each tree tries to correct the mistakes of the previous trees.

Gradient Boosting Machines

- A higher learning rate means each tree can make stronger corrections, allowing for more complex models.
- Adding more trees to the ensemble, which can be accomplished by increasing n_estimators, also increases the model complexity, as the model has more chances to correct mistakes on the training set.
- As both gradient boosting and random forests perform well on similar kinds of data, a common approach is to first try random forests, which work quite robustly.
- If random forests work well but prediction time is at a premium, or it is important to squeeze out the last percentage of accuracy from the machine learning model, moving to gradient boosting often helps.
- If you want to apply gradient boosting to a large-scale problem, it might be worth looking into the xgboost package and its Python interface,

Gradient Boosting Machines

- **Gradient boosted decision trees are among the most powerful and widely used models for supervised learning.**
- Their main drawback is that they require careful tuning of the parameters and may take a long time to train.
- Similarly to other tree-based models, the algorithm works well without scaling and on a mixture of binary and continuous features.
- As with other tree-based models, it also often does not work well on high-dimensional sparse data.

Gradient Boosting Machines

- The main parameters of gradient boosted tree models are the number of trees, n_estimators, and the learning_rate, which controls the degree to which each tree is allowed to correct the mistakes of the previous trees.
- These two parameters are highly interconnected, as a lower learning_rate means that more trees are needed to build a model of similar complexity.
- In contrast to random forests, where a higher n_estimators value is always better, increasing n_estimators in gradient boosting leads to a more complex model, which may lead to overfitting.
- A common practice is to fit n_estimators depending on the time and memory budget, and then search over different learning_rates.
- Another important parameter is max_depth (or alternatively max_leaf_nodes), to reduce the complexity of each tree.
- Usually max_depth is set very low for gradient boosted models, often not deeper than five splits.

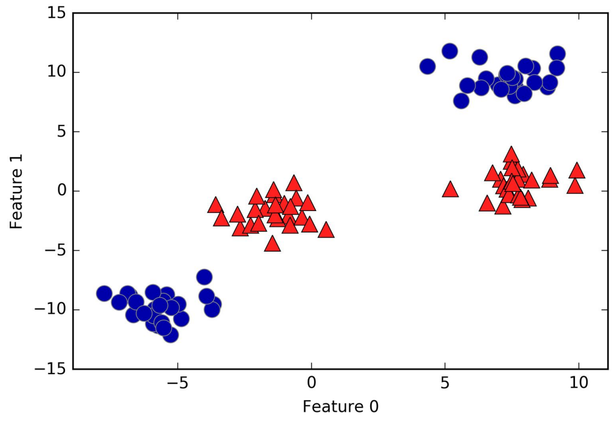
Gradient Boosting Machines - Demo

Kernelized Support Vector Machines

- We explored the use of linear support vector machines for classification in Linear models for classification
- Kernelized support vector machines (often just referred to as SVMs) are an extension that allows for more complex models that are not defined simply by hyperplanes in the input space.
- SVMs can be used for both classification and regression tasks.

Linear models and nonlinear features

- Linear models can be quite limiting in low-dimensional spaces, as lines and hyperplanes have limited flexibility.
- One way to make a linear model more flexible is by adding more features—for example, by adding interactions or polynomials of the input features.



Two-class classification dataset in which classes are not linearly separable