W4995 Applied Machine Learning
Fall 2021

Lecture 3

Dr. Vijay Pappu

Announcements

- 1st assignment due today at midnight (11:59 PM EST)
  - No late submissions are accepted
  - Please read all announcements posted by CAs
- Upcoming key dates:
  - HW2 posted on 10/06 (next Wed)
  - Project deliverable 1 due on 10/06 (next Wed)
- Projects groups
  - 1-2 teams have only 4 members

In today’s lecture, we will cover…

- Linear models for regression
- Linear models for classification

Linear models for regression

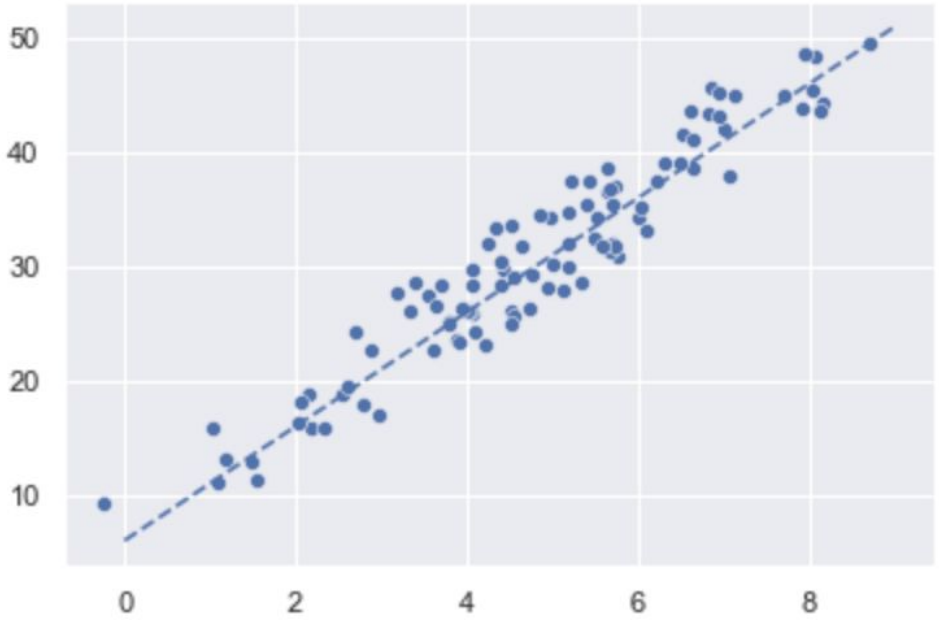
Linear models for regression

- Linear regression
- Ridge regression
- Lasso regression
- Elastic-net regression

Linear regression

Assumptions:

- Linearity
- Independence
- Homoscedasticity
- Normality



$$\hat{y} = X\vec{w} + \vec{b}$$

$$X \in \mathbb{R}^{m \times n}, \; w \in \mathbb{R}^{n}$$

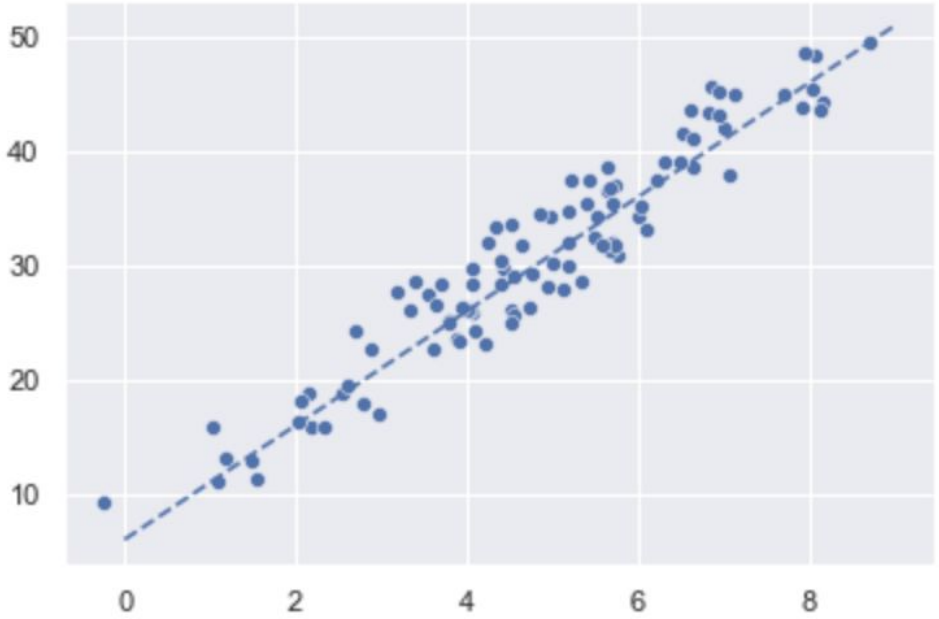
Linear regression

Least squares minimization

$$\underset{w}{Min} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2$$

$$w = (X^T X)^{-1} X^T y$$

Closed-form solution

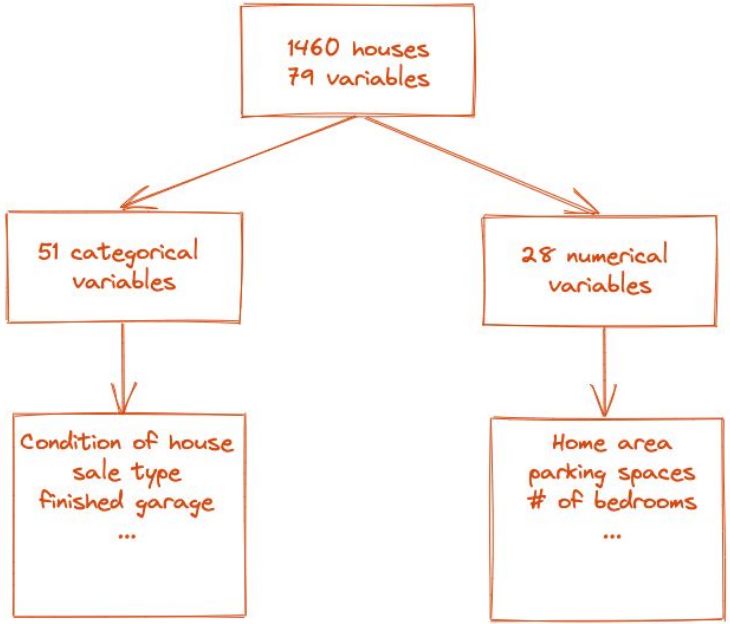


$$\hat{y} = X\vec{w} + \vec{b}$$

$$X \in \mathbb{R}^{m \times n}, \; w \in \mathbb{R}^n$$

Example dataset

Predict sale price of houses in Ames, Iowa and identify factors that impact the sale price.



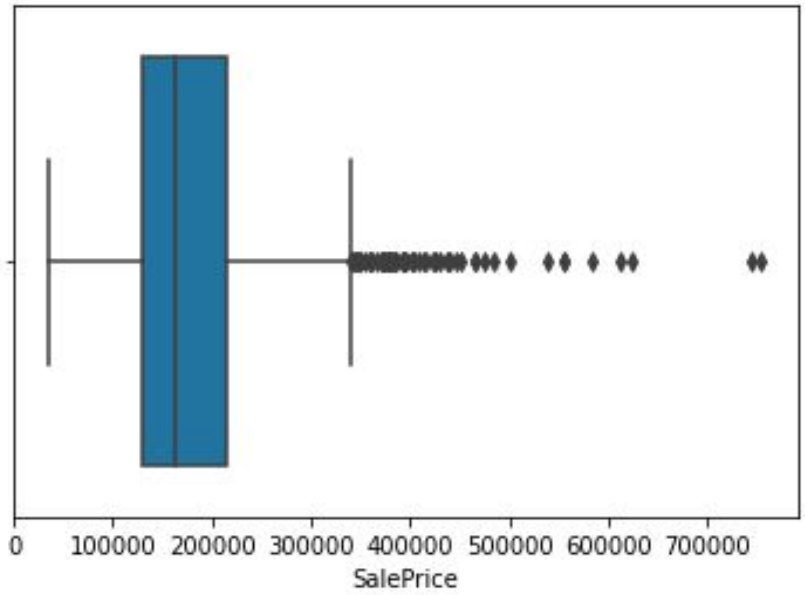
https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview/description

Example dataset

```
ohe_features = [
    "LotShape",
    "Street",
    "ExterCond",
    "OverallCond",
    "OverallQual",
    "Condition1",
    "Condition2",
    "Functional",
    "MSZoning",
    "FireplaceQu",
]
te_features = ["Neighborhood"]
num_features = [
    "BedroomAbvGr",
    "BsmtFinSF1",
    "BsmtFullBath",
    "EnclosedPorch",
    "GarageArea",
    "LotArea",
    "1stFlrSF",
    "2ndFlrSF",
    "TotalBsmtSF",
]
mixed_df = data_df[ohe_features + te_features + num_features]
target = data_df["SalePrice"]
print(mixed_df.shape)
```

```
(1460, 20)
```

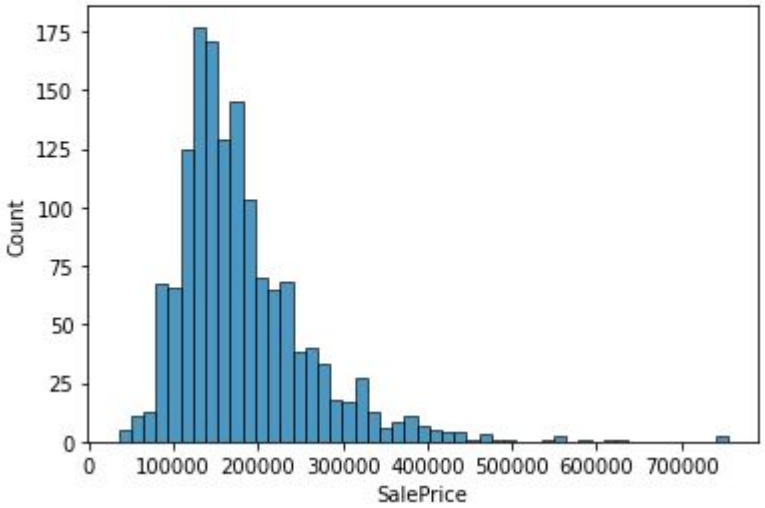
Target distribution



Target distribution

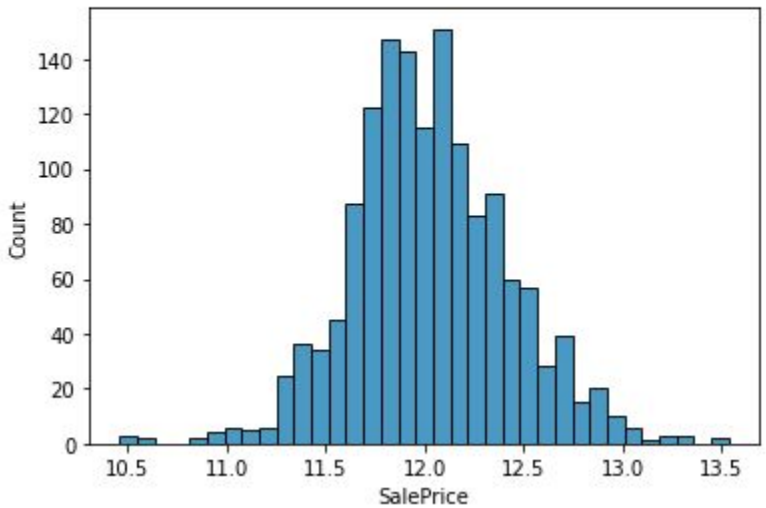
```
sns.histplot(target)
```

```
<AxesSubplot:xlabel='SalePrice', ylabel='Count'>
```




```
sns.histplot(np.log(target))
```

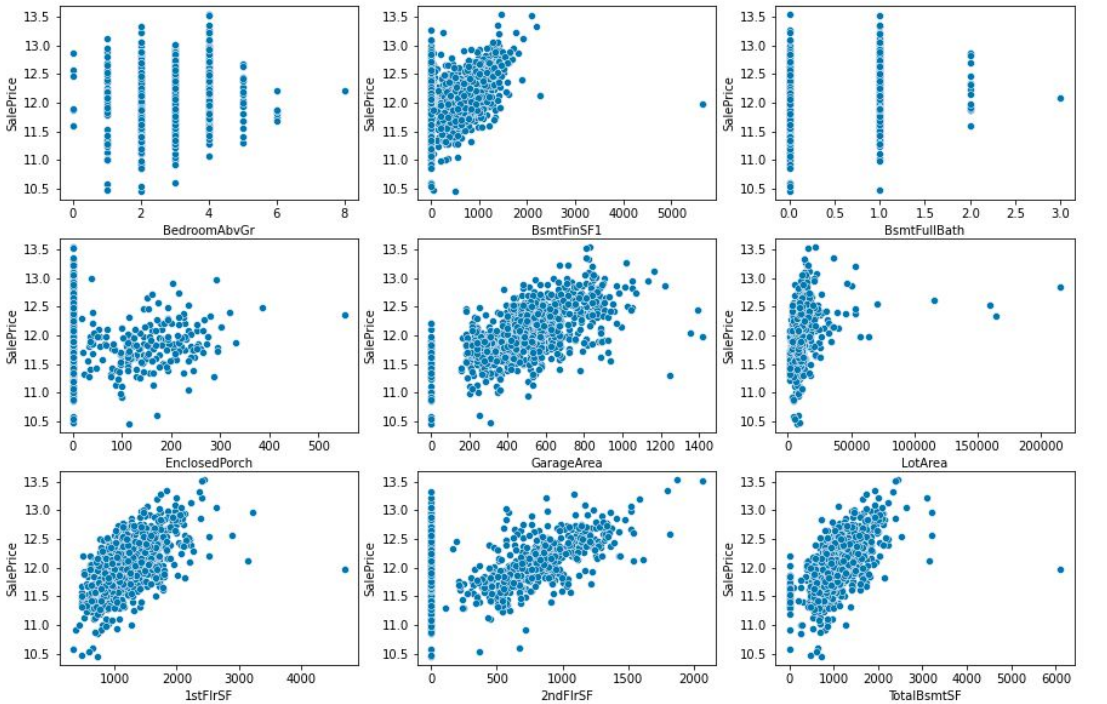
```
<AxesSubplot:xlabel='SalePrice', ylabel='Count'>
```



Let's use log(SalePrice) as our target

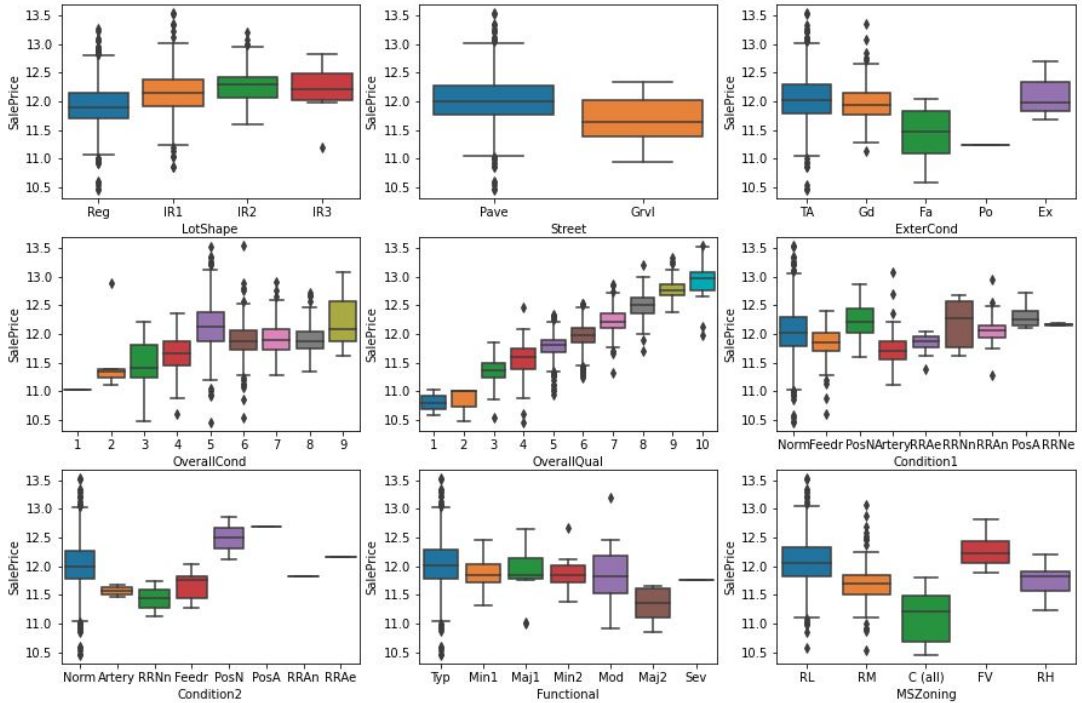
Numerical features v.s. target

```
fig, ax = plt.subplots(3, 3, figsize=(15, 10))
for var, subplot in zip(num_features, ax.flatten()):
    sns.scatterplot(x=var, y=target, data=mixed_df, ax=subplot)
```



Categorical features v.s. target

```
fig, ax = plt.subplots(3, 3, figsize=(15, 10))
for var, subplot in zip(ohe_features, ax.flatten()):
    sns.boxplot(x=var, y=target, data=mixed_df, ax=subplot)
```



Linear regression

```
ohe_features = [
    "LotShape",
    "Street",
    "ExterCond",
    "OverallCond",
    "OverallQual",
    "Condition1",
    "Condition2",
    "Functional",
    "MSZoning",
    "FireplaceQu",
]
te_features = ["Neighborhood"]
num_features = [
    "BedroomAbvGr",
    "BsmtFinSF1",
    "BsmtFullBath",
    "EnclosedPorch",
    "GarageArea",
    "LotArea",
    "1stFlrSF",
    "2ndFlrSF",
    "TotalBsmtSF",
]
mixed_df = data_df[ohe_features + te_features + num_features]
target = data_df["SalePrice"]
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)

preprocess = make_column_transformer((StandardScaler(), num_features),
                         (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                         (TargetEncoder(handle_unknown="value"), te_features),
                                  remainder="passthrough"
                        )
pipe = make_pipeline(preprocess, LinearRegression())
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

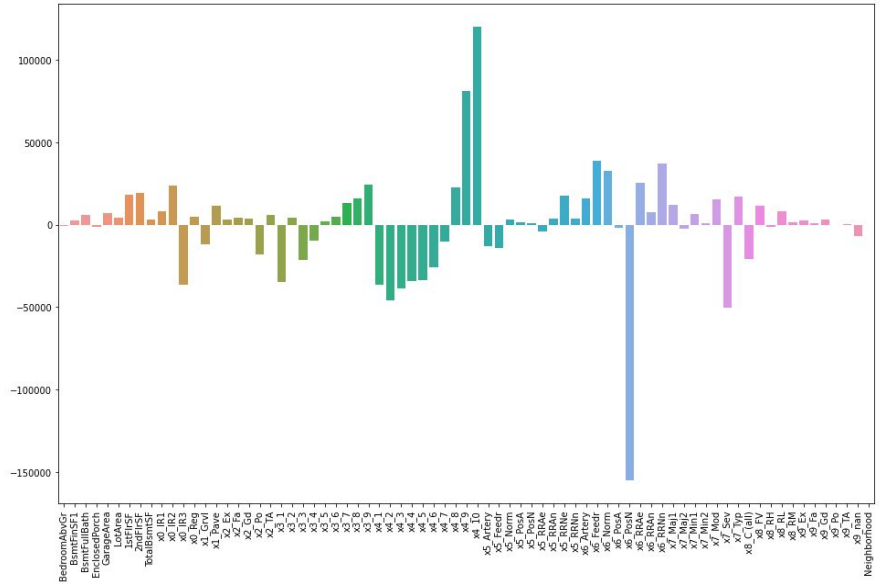
```
[0.89589344 0.82651304 0.87050954 0.70974225 0.36061735 0.86058542
 0.70811455 0.86788449 0.84712688 0.86075544]
0.7807742392302643
```

```
ohe_features = [
    "LotShape",
    "Street",
    "ExterCond",
    "OverallCond",
    "OverallQual",
    "Condition1",
    "Condition2",
    "Functional",
    "MSZoning",
    "FireplaceQu",
]
te_features = ["Neighborhood"]
num_features = [
    "BedroomAbvGr",
    "BsmtFinSF1",
    "BsmtFullBath",
    "EnclosedPorch",
    "GarageArea",
    "LotArea",
    "1stFlrSF",
    "2ndFlrSF",
    "TotalBsmtSF",
]
mixed_df = data_df[ohe_features + te_features + num_features]
target = np.log(data_df["SalePrice"])
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), num_features),
                         (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                         (TargetEncoder(handle_unknown="value"), te_features),
                                  remainder="passthrough"
                        )
pipe = make_pipeline(preprocess, LinearRegression())
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

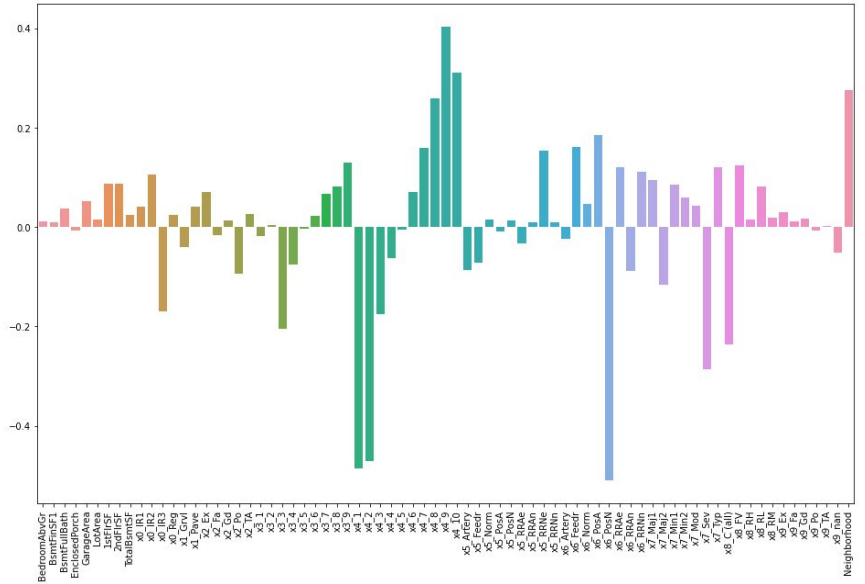
```
[0.89537986 0.83602003 0.85755909 0.77415228 0.49263643 0.86014632
 0.81151357 0.86437376 0.83683827 0.89543806]
0.8124057665890427
```

Linear regression - coefficients

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (16, 10))
ax = sns.barplot(x=feature_names, y=coefs)
ax.tick_params(axis='x', rotation=90)
```




```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (16, 10))
ax = sns.barplot(x=feature_names, y=coefs)
ax.tick_params(axis='x', rotation=90)
```

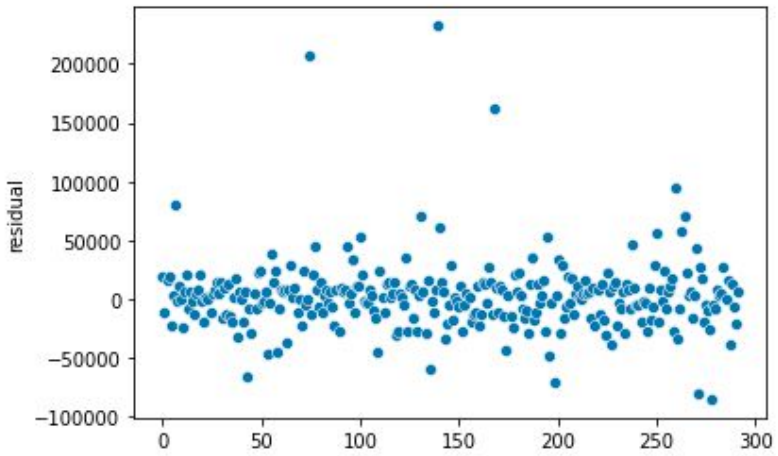


Target = log(SalePrice)

Linear regression - residual plots

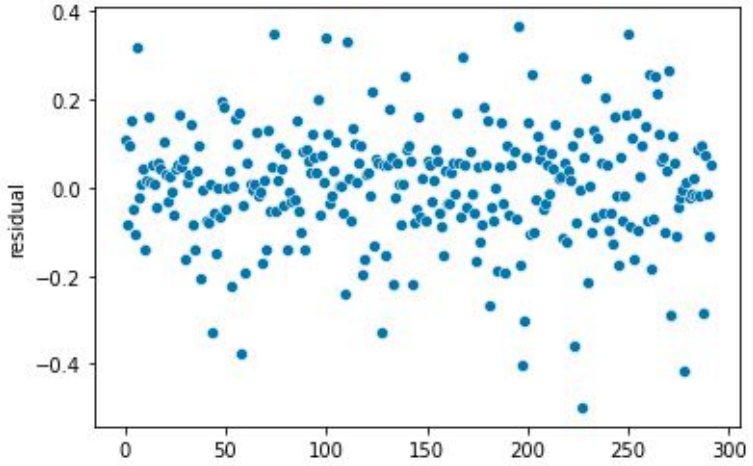
```
ax = sns.scatterplot(np.arange(len(test_y)),
                     (test_y - pipe.predict(test_X)))
ax.set_ylabel("residual")
```

```
Text(0, 0.5, 'residual')
```




```
ax = sns.scatterplot(np.arange(len(test_y)),
                     (test_y - pipe.predict(test_X)))
ax.set_ylabel("residual")
```

```
Text(0, 0.5, 'residual')
```



Target = log(SalePrice)

Ridge regression

$$\min_{w} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2$$

$$s.t. \sqrt{\sum_{i=1}^{n} w_i^2} \le c \quad \textbf{(L}_2\textbf{-norm)}$$

$$\Downarrow$$

$$\min_{w} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2 + \alpha \| w \|_2^2$$

$$w = (X^T X + \alpha I)^{-1} X^T y$$

Closed-form solution

$$\hat{y} = X\vec{w} + \vec{b}$$

$$X \in \mathbb{R}^{m \times n} \quad w \in \mathbb{R}^n$$

,

Ridge regression

```
from sklearn.linear_model import Ridge
ohe_features = [
    "LotShape",
    "Street",
    "ExterCond",
    "OverallCond",
    "OverallQual",
    "Condition1",
    "Condition2",
    "Functional",
    "MSZoning",
    "FireplaceQu",
]
te_features = ["Neighborhood"]
num_features = [
    "BedroomAbvGr",
    "BsmtFinSF1",
    "BsmtFullBath",
    "EnclosedPorch",
    "GarageArea",
    "LotArea",
    "1stFlrSF",
    "2ndFlrSF",
    "TotalBsmtSF",
]
mixed_df = data_df[ohe_features + te_features + num_features]
target = data_df["SalePrice"]
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)

preprocess = make_column_transformer((StandardScaler(), num_features),
                        (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                        (TargetEncoder(handle_unknown="value"), te_features),
                                    remainder="passthrough"
                       )
pipe = make_pipeline(preprocess, Ridge())
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
[0.59057126 0.42352026 0.56964418 0.46857538 0.49791895 0.48956644
 0.5742154  0.50807332 0.42764345 0.58332401]
0.5133052636248931
```

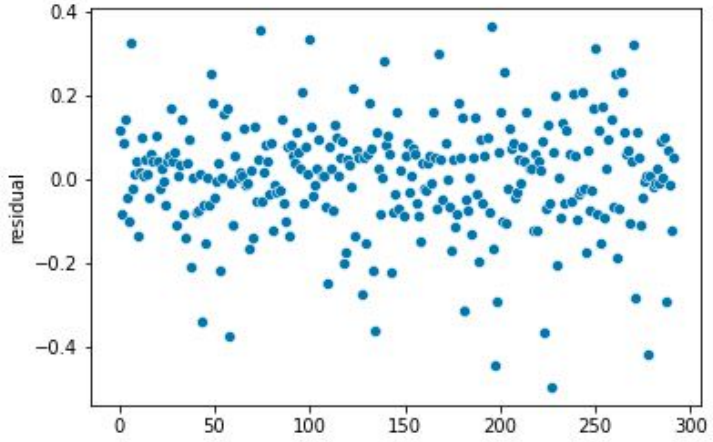
```
from sklearn.linear_model import Ridge
ohe_features = [
    "LotShape",
    "Street",
    "ExterCond",
    "OverallCond",
    "OverallQual",
    "Condition1",
    "Condition2",
    "Functional",
    "MSZoning",
    "FireplaceQu",
]
te_features = ["Neighborhood"]
num_features = [
    "BedroomAbvGr",
    "BsmtFinSF1",
    "BsmtFullBath",
    "EnclosedPorch",
    "GarageArea",
    "LotArea",
    "1stFlrSF",
    "2ndFlrSF",
    "TotalBsmtSF",
]
mixed_df = data_df[ohe_features + te_features + num_features]
target = np.log(data_df["SalePrice"])
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)

preprocess = make_column_transformer((StandardScaler(), num_features),
                        (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                        (TargetEncoder(handle_unknown="value"), te_features),
                                    remainder="passthrough"
                       )
pipe = make_pipeline(preprocess, Ridge())
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
[0.89387861 0.85150629 0.86695871 0.7847623  0.53154992 0.85641226
 0.85830536 0.86527927 0.83872405 0.89674621]
0.8244122971907594
```

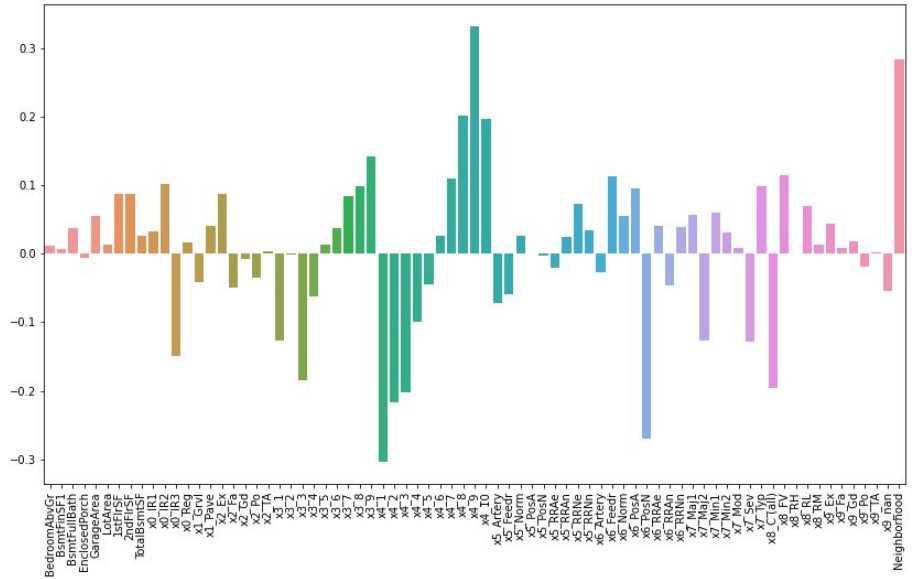
Ridge regression

```
pipe.fit(dev_X, dev_y)
ax = sns.scatterplot(np.arange(len(test_y)),
                     (test_y - pipe.predict(test_X)))
ax.set_ylabel("residual")
```

```
Text(0, 0.5, 'residual')
```




```
pipe.fit(dev_X, dev_y)
ohe_feature_names = preprocess.named_transformers_["onehotencoder"].get_feature_names().tolist()
te_feature_names = preprocess.named_transformers_["targetencoder"].get_feature_names()
feature_names = num_features + ohe_feature_names + te_feature_names
coefs = pipe.named_steps["ridge"].coef_
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (14, 8))
ax = sns.barplot(x=feature_names, y=coefs)
ax.tick_params(axis='x', rotation=90)
```



Ridge regression - hyperparameter tuning

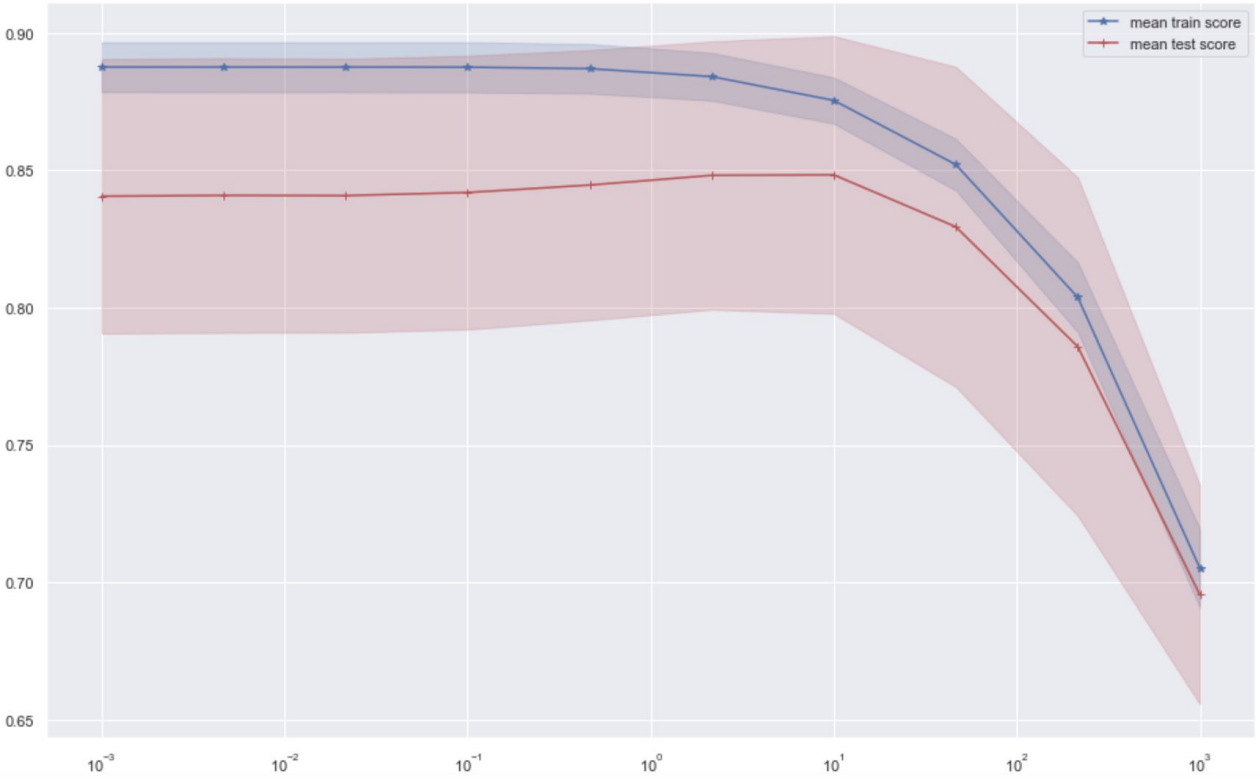
```
preprocess = make_column_transformer((StandardScaler(), num_features),
                        (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                        (TargetEncoder(handle_unknown="value"), te_features),
                                    remainder="passthrough"
                       )
pipe = make_pipeline(preprocess,
                     GridSearchCV(Ridge(),
                                  param_grid = [{"alpha":np.logspace(-3, 3, 10)}],
                                  return_train_score=True))
pipe.fit(mixed_df, target)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Best score: 0.8483693747019642
Best alpha: {'alpha': 10.0}
Test score: 0.9112277632865977
```

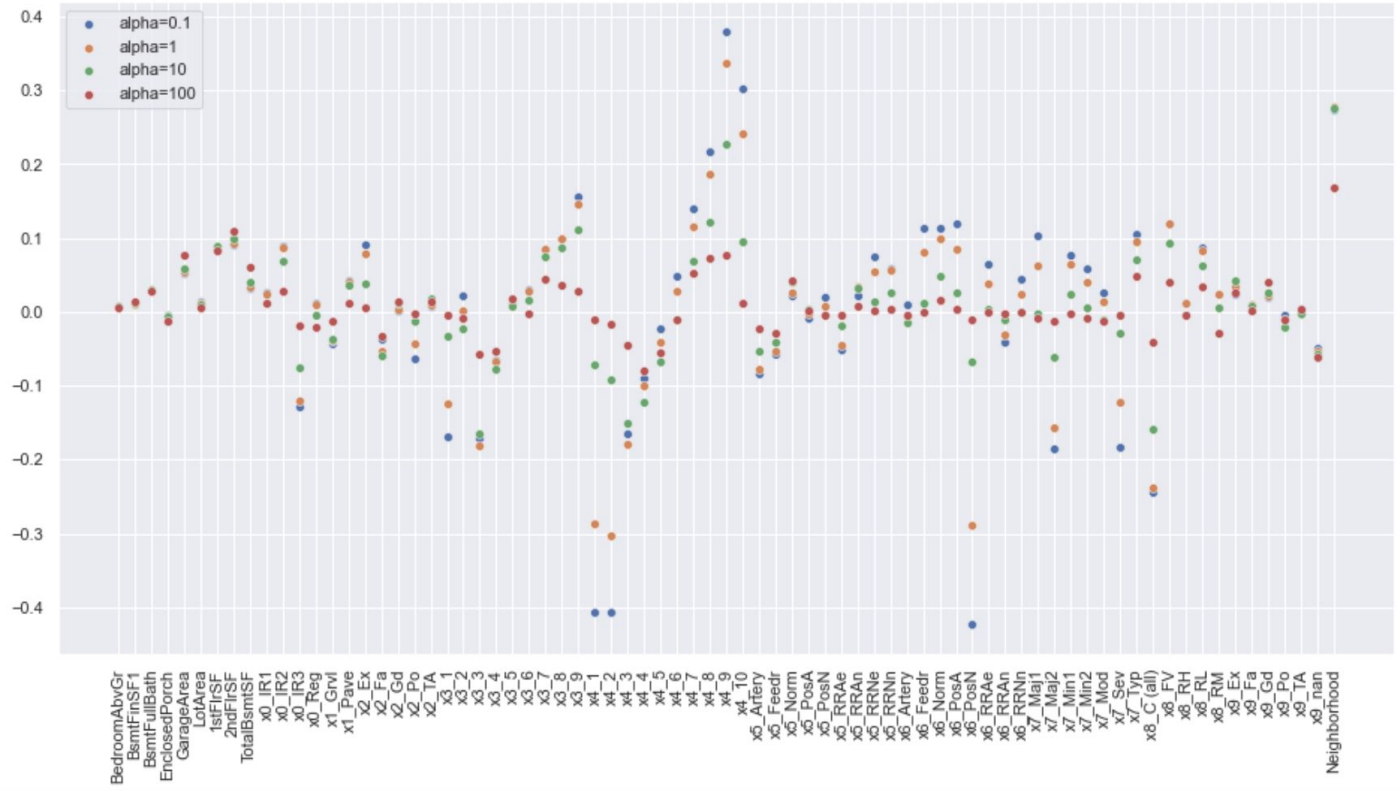
Ridge regression - hyperparameter tuning

```
grid_search_results = pipe.named_steps["gridsearchcv"]
mean_test_score = grid_search_results.cv_results_["mean_test_score"]
std_test_score = grid_search_results.cv_results_["std_test_score"]
mean_train_score = grid_search_results.cv_results_["mean_train_score"]
std_train_score = grid_search_results.cv_results_["std_train_score"]
grid_search_results = pipe.named_steps["gridsearchcv"]
sns.set()
fig = plt.figure(figsize=(16,10))
ax = fig.add_subplot(1, 1, 1)
plt.plot(alpha_params, mean_train_score, 'b*-', label='mean train score')
plt.fill_between(alpha_params, mean_train_score - std_train_score,
                 mean_train_score + std_train_score, color='b', alpha=0.2)
plt.plot(alpha_params, mean_test_score, 'r+-', label='mean test score')
plt.fill_between(alpha_params, mean_test_score - std_test_score,
                 mean_test_score + std_test_score, color='r', alpha=0.2)
plt.legend()
ax.set_xscale('log')
plt.show()
```

Ridge regression - hyperparameter tuning



Ridge regression - coefficients v.s. alpha



Lasso regression

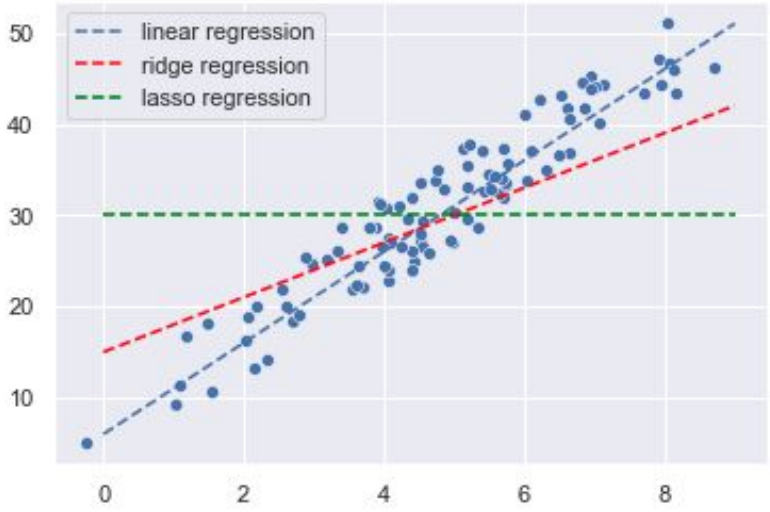
$$\underset{w}{Min} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2$$

$$s.t. \sum_{i=1}^{n} |w_i| \leq c \quad \text{(L}_1\text{-norm)}$$

$$\Downarrow$$

$$\underset{w}{Min} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2 + \alpha \| w \|_1$$

No closed-form solution



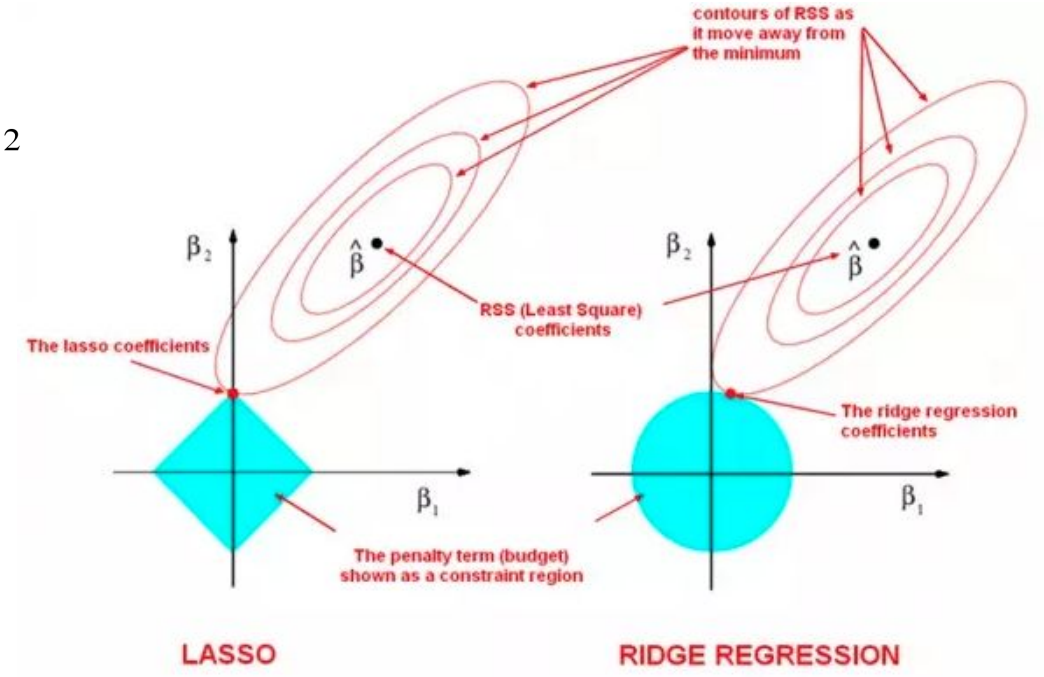
$$\hat{y} = X\vec{w} + \vec{b}$$

$$X \in \mathbb{R}^{m \times n} \quad w \in \mathbb{R}^n$$

Lasso regression promotes feature selection by setting some coefficients to zero

Ridge regression v.s. Lasso regression

$$\min_{w} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2 \quad s.t. \sum_{i=1}^{n} |w_i| \le c$$



$$\min_{w} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2 \quad s.t. \sqrt{\sum_{i=1}^{n} w_i^2} \le c$$

https://www.quora.com/How-would-you-describe-the-difference-between-linear-regression-lasso-regression-and-ridge-regression

Lasso regression - example

```
from sklearn.linear_model import Lasso
mixed_df = data_df[ohe_features + te_features + num_features]
target = np.log(data_df["SalePrice"])
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)

preprocess = make_column_transformer((StandardScaler(), num_features),
                         (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                         (TargetEncoder(handle_unknown="value"), te_features),
                                      remainder="passthrough"
                        )
pipe = make_pipeline(preprocess, Lasso(alpha=1.0))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
[-1.51312578e-03 -3.49328192e-03 -2.93540165e-02 -3.56696218e-03
 -1.46702996e-02 -9.79493587e-03 -4.55509622e-03 -3.73018028e-04
 -9.57262546e-05 -7.04991496e-03]
-0.007446637736710971
```

Lasso regression - example

```
from sklearn.linear_model import Lasso
mixed_df = data_df[ohe_features + te_features + num_features]
target = np.log(data_df["SalePrice"])
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)

preprocess = make_column_transformer((StandardScaler(), num_features),
                        (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                        (TargetEncoder(handle_unknown="value"), te_features),
                                    remainder="passthrough"
                       )
pipe = make_pipeline(preprocess, Lasso(alpha=0.1))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

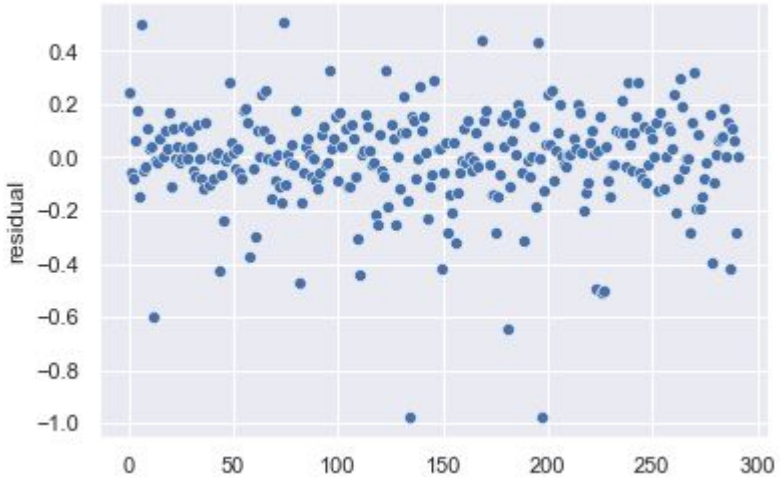
```
[0.52802342 0.5065579  0.53182997 0.44564904 0.33896109 0.4322538
 0.48875767 0.48883398 0.43652834 0.49754664]
0.46949418651484065
```

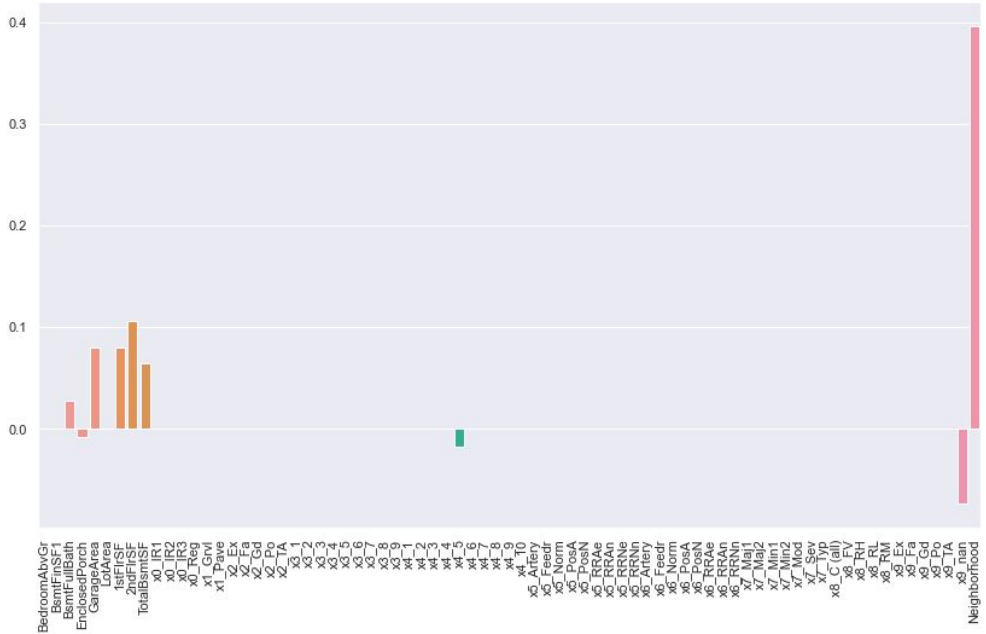
Lasso regression - example

```
from sklearn.linear_model import Lasso
mixed_df = data_df[ohe_features + te_features + num_features]
target = np.log(data_df["SalePrice"])
dev_X, test_X, dev_y, test_y = train_test_split(mixed_df, target,
                                                test_size=0.2, random_state=42)

preprocess = make_column_transformer((StandardScaler(), num_features),
                         (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                         (TargetEncoder(handle_unknown="value"), te_features),
                                      remainder="passthrough"
                        )
pipe = make_pipeline(preprocess, Lasso(alpha=0.01))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
[0.8366061  0.83438682 0.82532131 0.70633451 0.42958249 0.77676291
 0.820875   0.78605352 0.77910051 0.82373417]
0.7618757335492352
```

Lasso regression



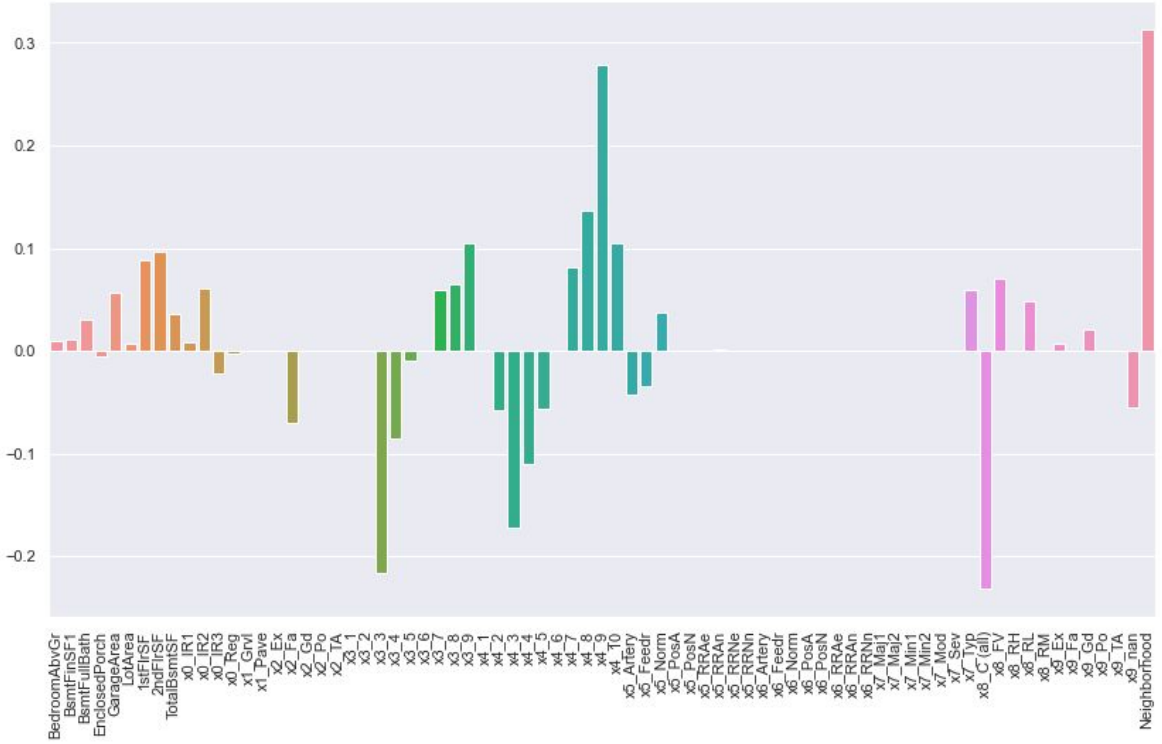


Lasso regression - hyperparameter tuning

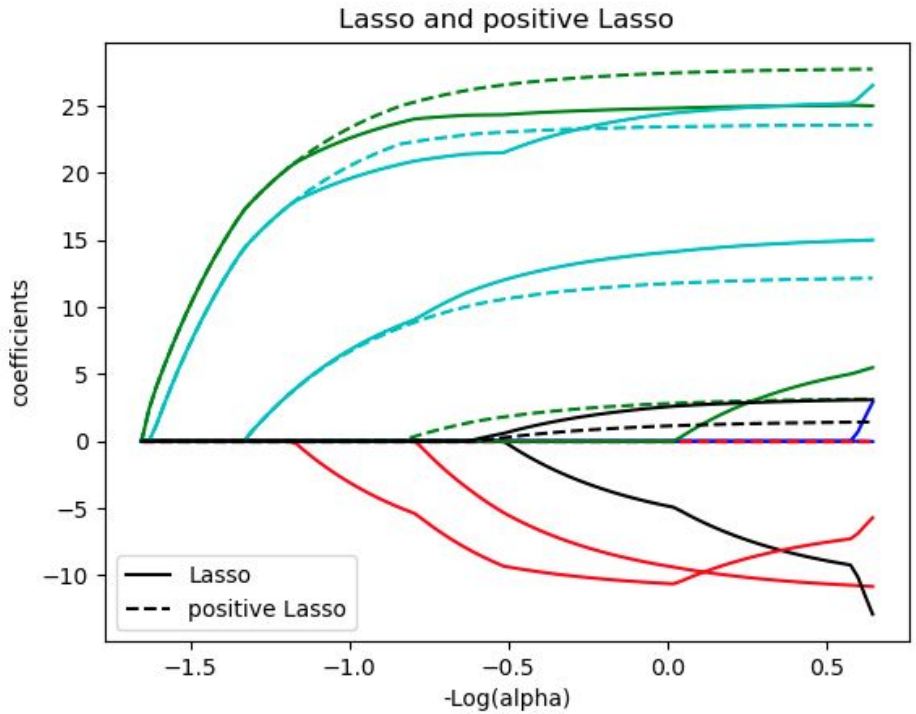
```
preprocess = make_column_transformer((StandardScaler(), num_features),
                        (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                        (TargetEncoder(handle_unknown="value"), te_features),
                                    remainder="passthrough"
                       )
pipe = make_pipeline(preprocess,
                     GridSearchCV(Lasso(),
                                  param_grid = [{"alpha":np.logspace(-4, 2, 30)}],
                                  return_train_score=True))
pipe.fit(mixed_df, target)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Best score: 0.8488535219688801
Best alpha: {'alpha': 0.0006723357536499335}
Test score: 0.911783413368781
```

Lasso regression - hyperparameter tuning



Lasso regression - regularization path



https://bit.ly/3CPruz8

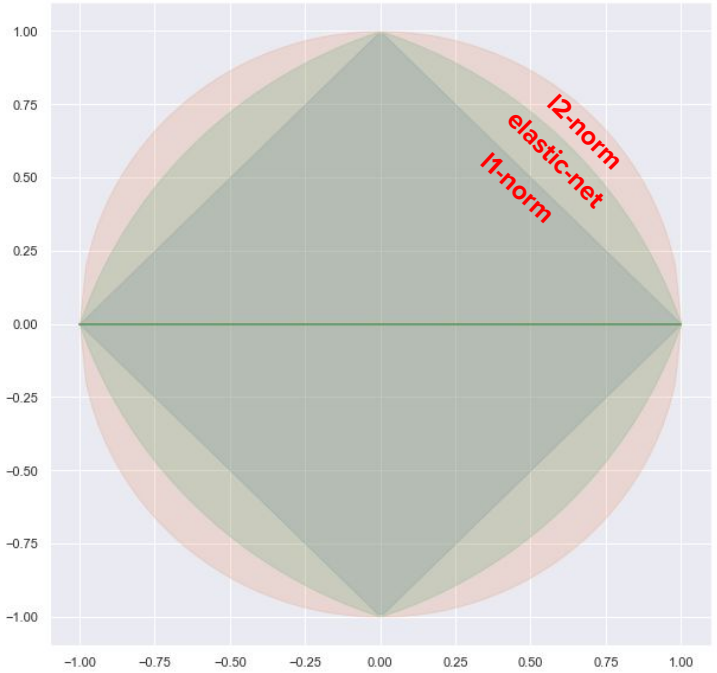
Elastic-Net regression

$$\underset{w}{Min} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2$$

$$s.t. \ \lambda \sum_{i=1}^{n} |w_i| + (1-\lambda) \sum_{i=1}^{n} w_i^2 \leq c$$

$$\underset{w}{Min} \sum_{i=1}^{m} \left(\hat{y}_i - y_i \right)^2 + \alpha \left(\lambda \sum_{i=1}^{n} \| w \|_1 + (1-\lambda) \sum_{i=1}^{n} \| w \|_2^2 \right) \leq c$$

No closed-form solution



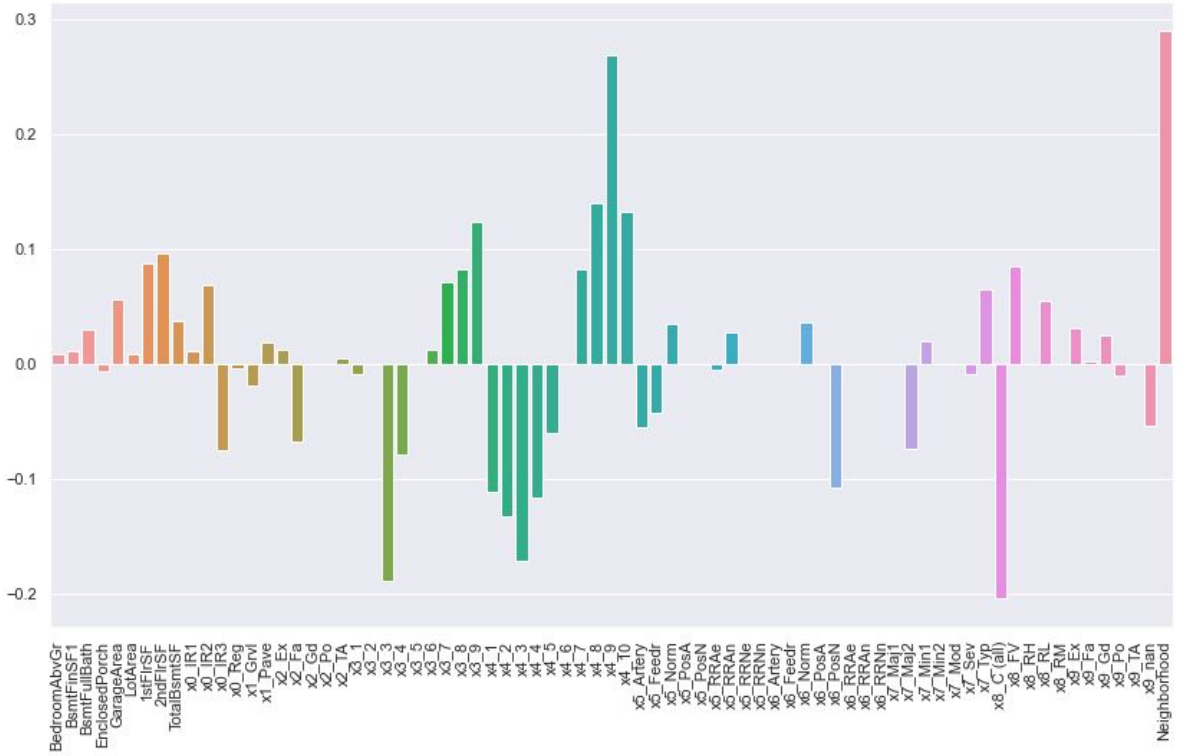
,

Elastic-net regression - hyperparameter tuning

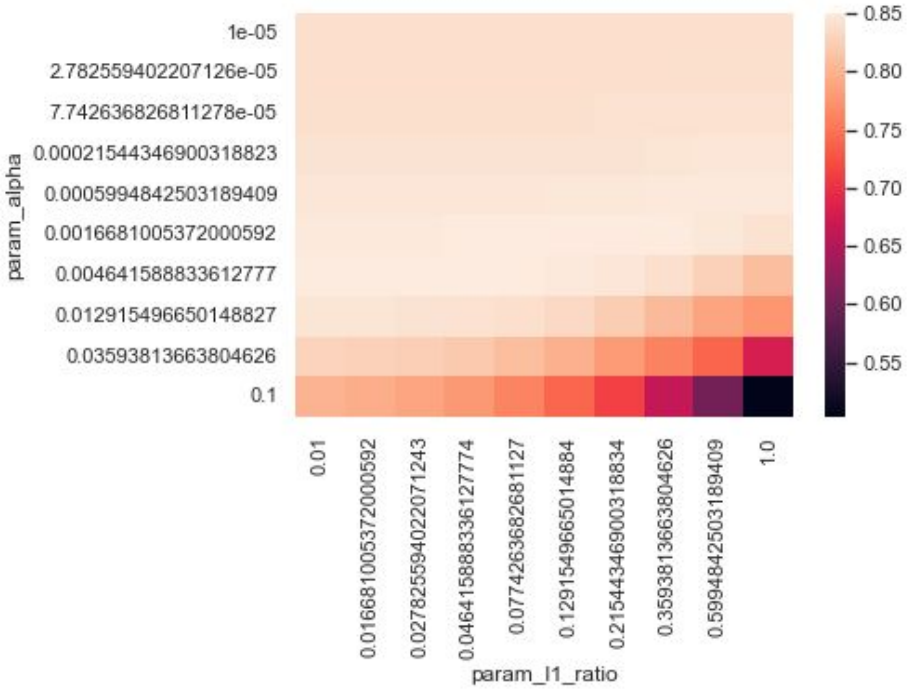
```
from sklearn.linear_model import ElasticNet
preprocess = make_column_transformer((StandardScaler(), num_features),
                        (OneHotEncoder(handle_unknown="ignore"), ohe_features),
                        (TargetEncoder(handle_unknown="value"), te_features),
                                    remainder="passthrough"
                       )
pipe = make_pipeline(preprocess,
                     GridSearchCV(ElasticNet(),
                                  param_grid = [{"alpha":np.logspace(-4, 2, 30),
                                                 "l1_ratio":np.logspace(-2, 0, 10)}],
                                  return_train_score=True))
pipe.fit(mixed_df, target)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Best score: 0.8509003475854822
Best alpha: {'alpha': 0.0028072162039411755, 'l1_ratio': 0.0774263682681127}
Test score: 0.914450036288246
```

Elastic-net regression - hyperparameter tuning



Elastic-net regression - hyperparameter tuning



Questions?

Let’s take a 10 min break!

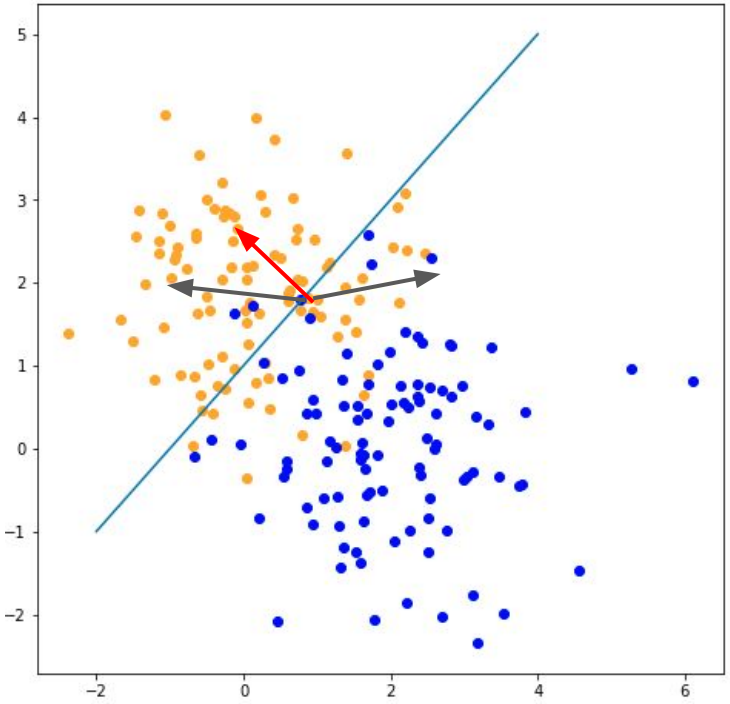
Linear models for classification

Linear models for Classification

- Binary classification
  - Logistic Regression
  - Regularized Logistic Regression
  - Hard Margin SVMs
  - Soft Margin SVMs
  - *Kernel SVMs
- Multi-class classification
  - One v.s. One & One v.s. Rest
  - Multinomial Logistic Regression

Binary classification

A simple linear model for classification



$$\hat{y} = sign(\, w^T x \, + \, b)$$

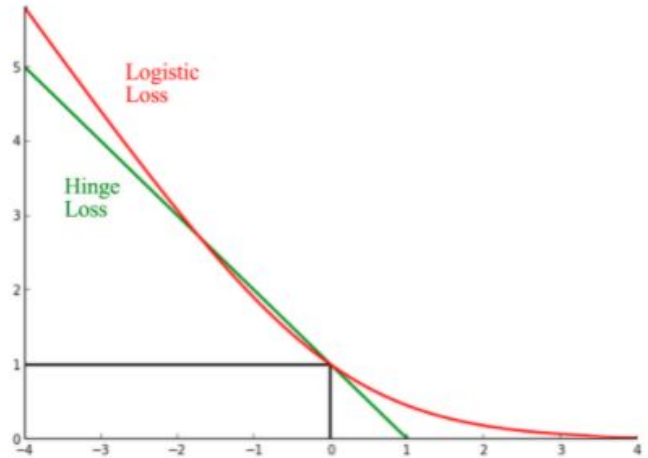
How do we estimate *w* and b?

Minimizing Loss Function

- **0-1 loss function:**

$$\underset{w,\ b}{Min} \sum_{i=1}^{m} 1_{y_i \neq sign(w^T x + b)}$$

- **Other loss functions:**
  - **Hinge loss:** $\underset{w,\ b}{Min} \sum_{i=1}^{m} Max\left(0,\ 1 - y_i w^T x_i\right)$
  - **Log loss:** $\underset{w,\ b}{Min} \left(- \sum_{i=1}^{m} y_i \log\left(p_i\right) + \left(1 - y_i\right) \log\left(1 - p_i\right) \right)$



Logistic Regression

$$\log\left(\frac{p(y=1|x)}{p(y=-1|x)}\right) = w^T x + b$$

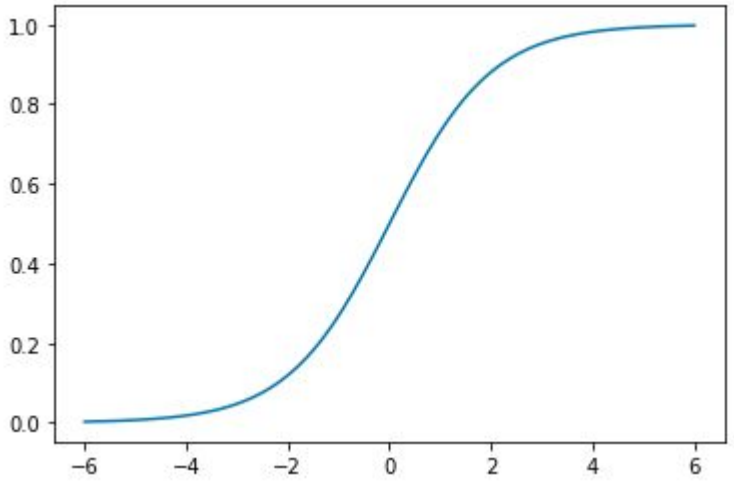
Log odds ratio

$$p(y=1|x) = \frac{1}{1 + exp(-(w^T x + b))}$$

$$Log\ Likelihood\ (LL) = \sum_{i=1}^{m} y_i \log(p_i) + (1-y_i)\log(1-p_i)$$

$$\underset{w,\ b}{Min}\ (-LL)$$

Negative Log likelihood (NLL)



Logistic Regression - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, LogisticRegression(C=1e6))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[0.97826087 0.95652174 0.97826087 0.91304348 1.         1.
 0.95555556 0.97777778 0.93333333 0.95555556]
0.9648309178743961
```

Regularized Logistic Regression

$$\underset{w,\,b}{Min}\left(-\sum_{i=1}^{m} y_i \log(p_i) + (1-y_i)\log(1-p_i)\right) + \alpha \| w \|_1$$

Lasso logistic regression

$$\underset{w,\,b}{Min}\left(-\sum_{i=1}^{m} y_i \log(p_i) + (1-y_i)\log(1-p_i)\right) + \alpha \| w \|_2^2$$

Ridge logistic regression

$$\underset{w,\,b}{Min}\left(-\sum_{i=1}^{m} y_i \log(p_i) + (1-y_i)\log(1-p_i)\right) + \alpha\lambda \| w \|_1 + \alpha(1-\lambda) \| w \|_2^2$$

Elastic-net logistic regression

Lasso Logistic Regression - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, LogisticRegression('l1', C=0.1, solver='liblinear'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[1.         0.97826087 0.97826087 0.93478261 0.97826087 0.97777778
 0.97777778 0.95555556 0.95555556 0.95555556]
0.9691787439613526
```

Lasso Logistic Regression - hyperparameter tuning

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
pipe = make_pipeline(preprocess,
                     GridSearchCV(LogisticRegression(),
                                  param_grid = [{"penalty":['l1'],
                                                 "solver":["liblinear"],
                                                 "C":np.logspace(-3, 3, 10)}],
                                  return_train_score=True))
pipe.fit(dev_X, dev_y)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
Best score: 0.9758241758241759
Best alpha: {'C': 2.154434690031882, 'penalty': 'l1', 'solver': 'liblinear'}
Test score: 0.9649122807017544
```

Ridge Logistic Regression - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, LogisticRegression('l2', C=0.1, solver='liblinear'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[1.         0.97826087 0.97826087 0.95652174 0.97826087 1.
 0.97777778 0.97777778 0.97777778 0.97777778]
0.9802415458937197
```

Ridge Logistic Regression - hyperparameter tuning

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
pipe = make_pipeline(preprocess,
                     GridSearchCV(LogisticRegression(),
                                  param_grid = [{"penalty":['l2'],
                                                 "solver":["liblinear"],
                                                 "C":np.logspace(-3, 3, 10)}],
                                  return_train_score=True))
pipe.fit(dev_X, dev_y)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
Best score: 0.9780219780219781
Best alpha: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Test score: 0.9912280701754386
```

Elastic Net Logistic Regression - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, LogisticRegression('elasticnet', C=0.1,
                                                    solver='saga', l1_ratio=0.5))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

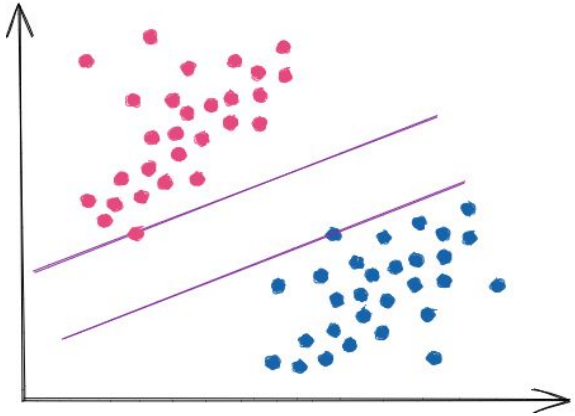
```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[1.         0.97826087 0.97826087 0.95652174 0.97826087 1.
 0.97777778 0.97777778 0.97777778 0.91111111]
0.9735748792270531
```

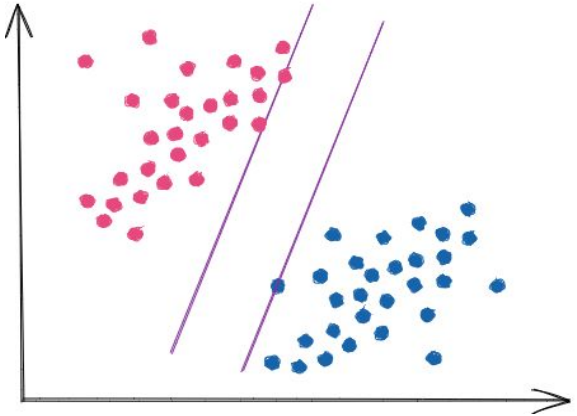
Elastic Net Logistic Regression - hyperparameter tuning

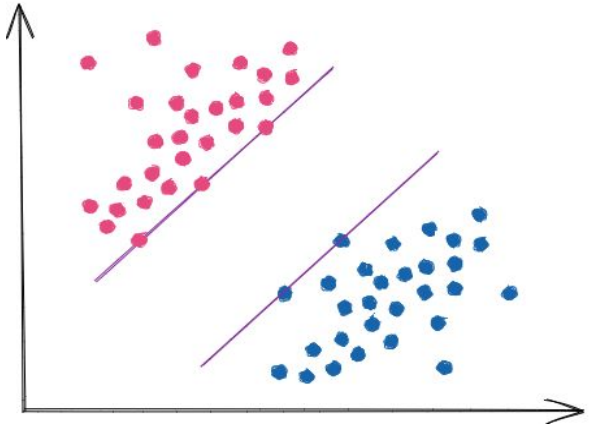
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
pipe = make_pipeline(preprocess,
                     GridSearchCV(LogisticRegression(),
                                  param_grid = [{"penalty":['elasticnet'],
                                                 "solver":["saga"],
                                                 "C":np.logspace(-3, 3, 10),
                                                 "l1_ratio": np.linspace(0, 1, 10)}],
                                  return_train_score=True))
pipe.fit(dev_X, dev_y)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
Best score: 0.9758241758241759
Best alpha: {'C': 0.46415888336127775, 'l1_ratio': 0.0, 'penalty': 'elasticnet', 'solver': 'saga'}
Test score: 0.9824561403508771
```

Max-margin classifiers

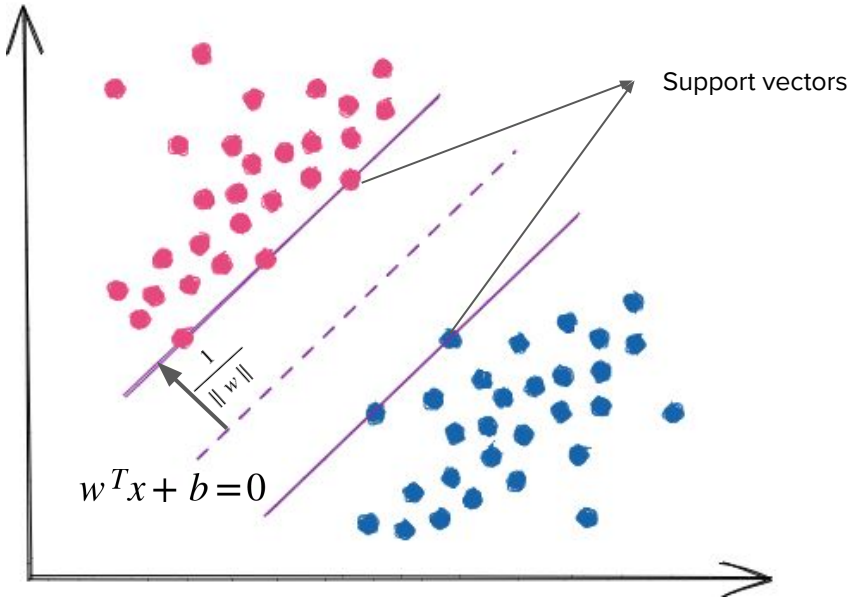




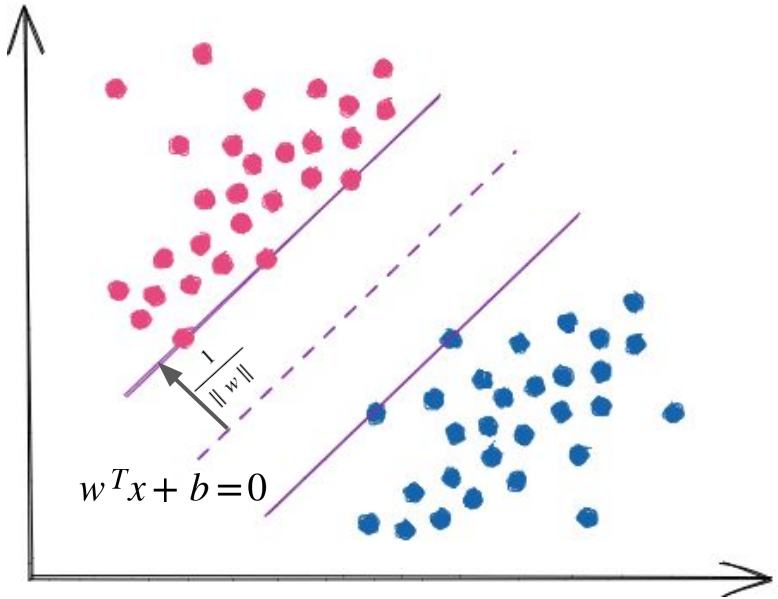


Support Vector Machines (SVMs)

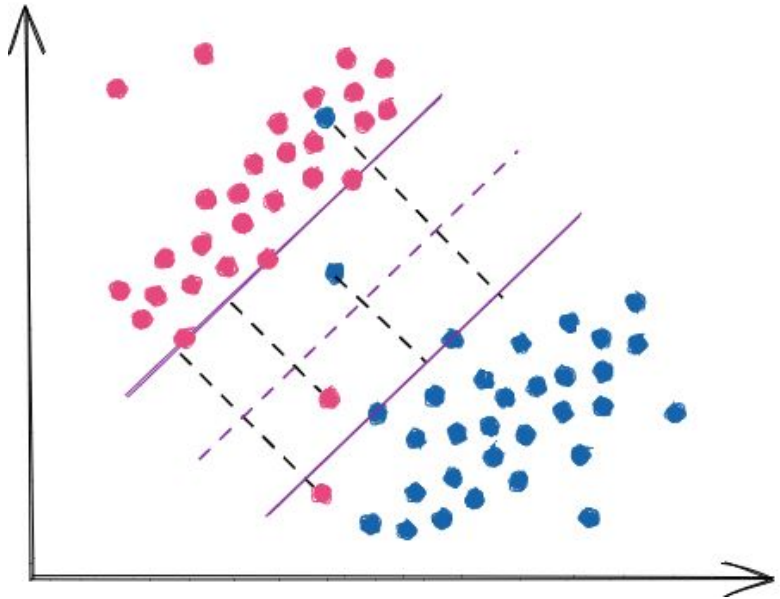
- Convex (quadratic) optimization
- Theoretical properties
- Easily extensible to non-linear case



Support Vector Machines (SVMs)



Hard-margin SVMs
(linearly separable)



Soft-margin SVMs
(linearly non-separable)

Hard-margin SVMs (Primal)

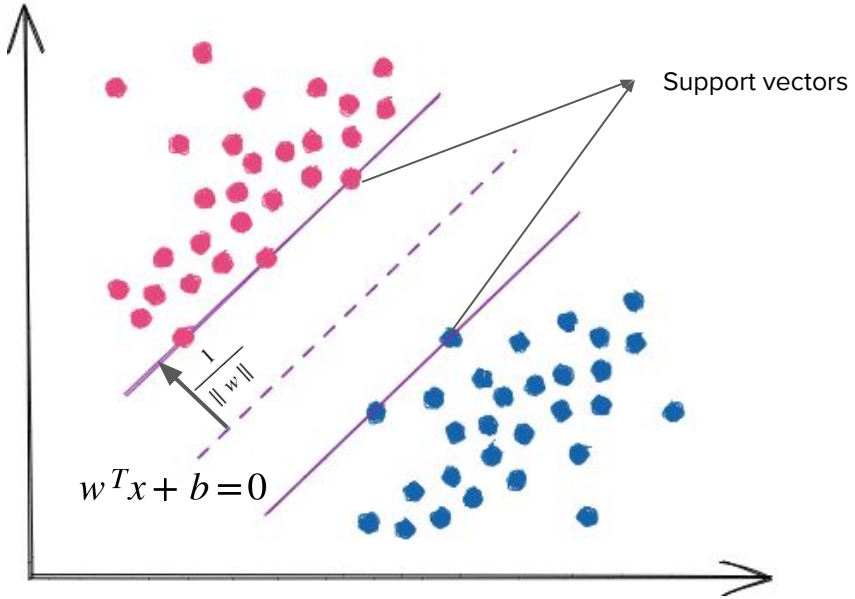
$$\underset{w,b}{Min}\ \frac{1}{2}\| w \|_2^2$$

$$s.t.\ \ y_i\left(w^T x_i + b \right) \geq 1$$

$$y_i \in \{-1, 1\}$$

Convex optimization

Prediction: $sign(w^T x + b)$



Hard-margin SVMs (Dual)

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \right)$$

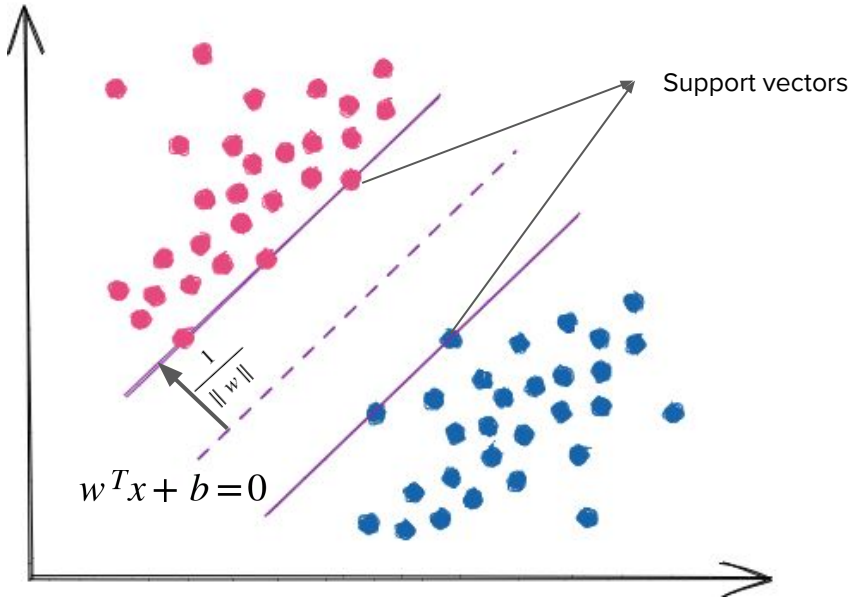
$$s.t. \sum_{i=1}^{m} \alpha_j y_j = 0$$

$$0 \leq \alpha_i \; \forall i$$

Convex optimization

Prediction: $sign\left(\sum_i \alpha_i y_i \left(x \cdot x_i \right) + b \right)$

support vectors



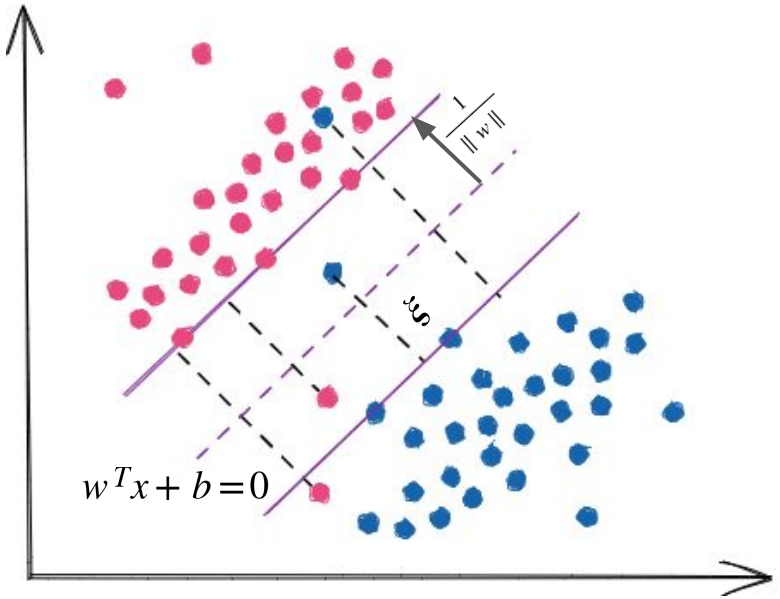
Soft-margin SVMs (Primal)

$$\underset{w,b}{Min}\ \frac{1}{2}\| w\|_2^2 + C\sum_{i=1}^{m}\xi_i$$

$$s.t.\ \ y_i\left(w^T x_i + b\right) \geq 1 - \xi_i$$

$$y_i \in \{-1, 1\}$$

Prediction: $sign(w^T x + b)$

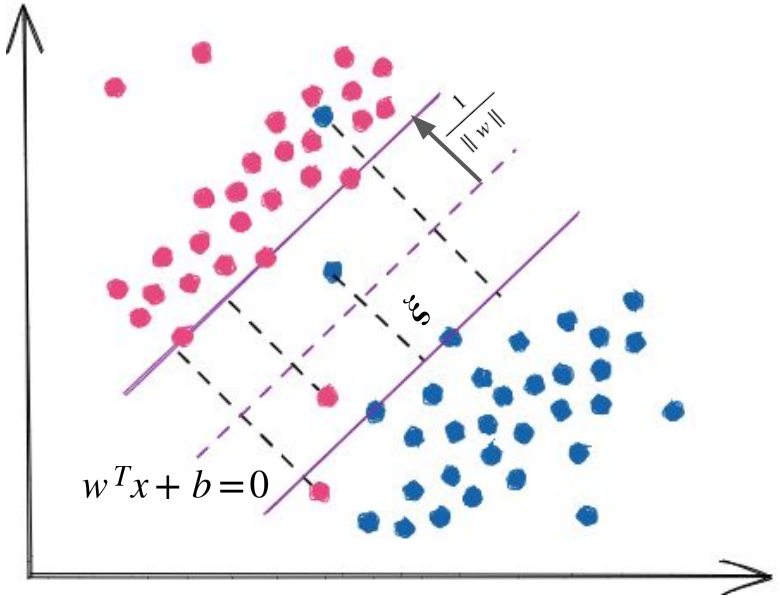


Soft-margin SVMs (Primal)

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m}\left(Max\left(0,\ 1-y_i\left(w^T x_i+b\right)\right)\right)\ +\ \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

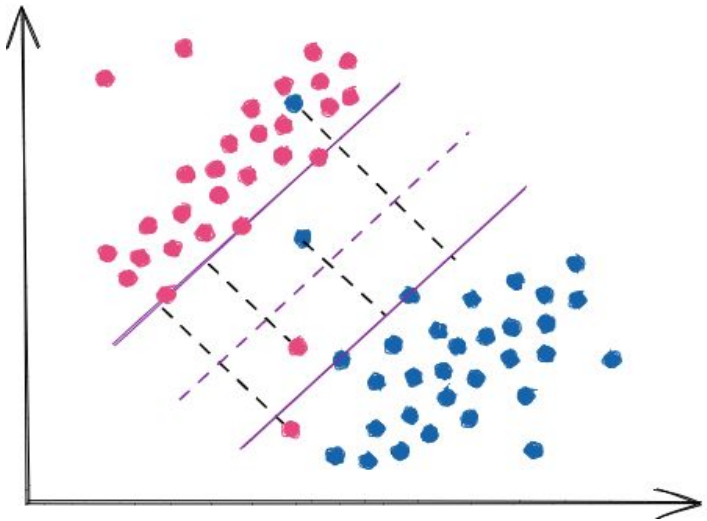
Hinge loss

Prediction: $sign(w^T x+b)$

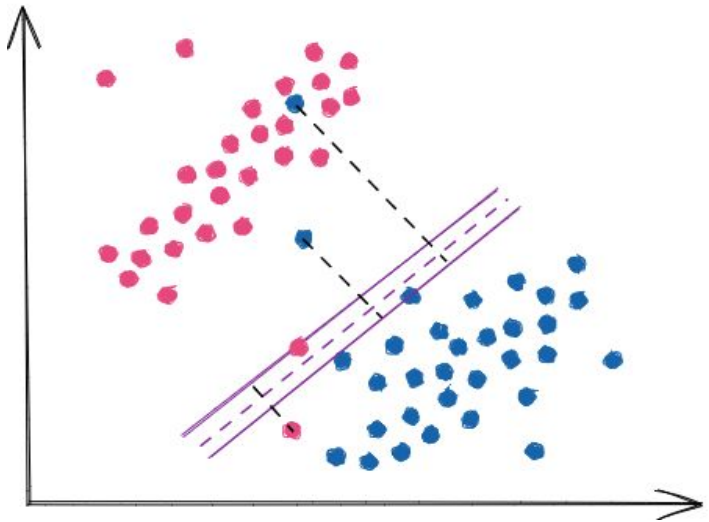


Soft-margin SVMs (Primal)

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m}\left(Max\left(0,\ 1-y_i\left(w^T x_i+b\right)\right)\right)\ +\ \frac{1}{2}\|\ \boldsymbol{w}\|_2^2$$



C = 0.01



C = 100

Smaller values of C leads to wider margins (and vice-versa)

Soft-margin SVMs (Dual)

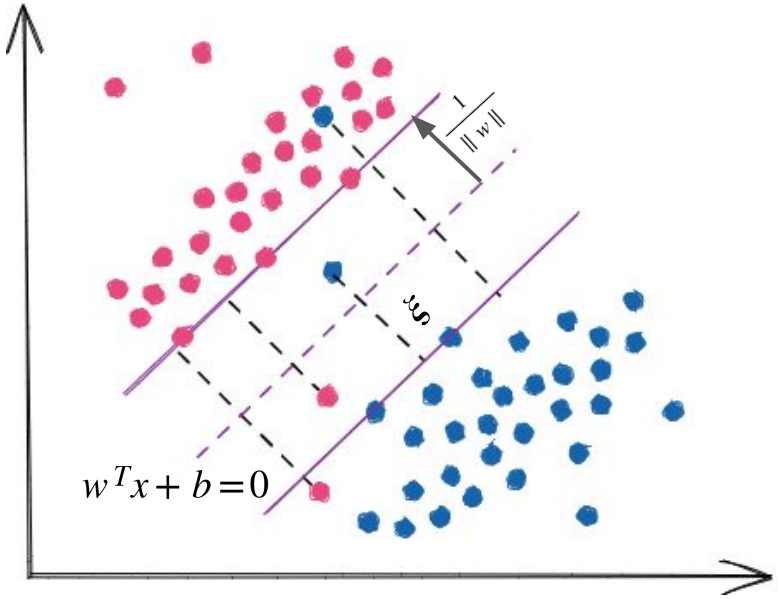
$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \right)$$

$$s.t. \sum_{i=1}^{m} \alpha_j y_j = 0$$

$$0 \leq \alpha_i \leq C \ \forall i$$

Prediction: $sign\left(\sum_i \alpha_i y_i \left(x \cdot x_i \right) + b \right)$

support vectors



Soft-margin SVMs - variants

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m}\left(Max\left(0,\ 1-y_i\left(w^T x_i+b\right)\right)\right)\ +\ \|\boldsymbol{w}\|_1$$

l1-norm

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m}\left(Max\left(0,\ 1-y_i\left(w^T x_i+b\right)\right)\right)\ +\ \alpha\lambda\|\boldsymbol{w}\|_1+\alpha(1-\lambda)\|\boldsymbol{w}\|_2^2$$

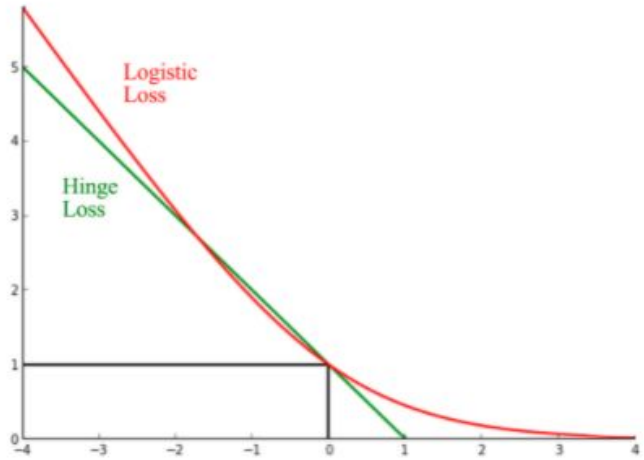
Elastic net

Logistic Regression v.s. SVMs

SVMs:

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m}\left(Max\left(0,\ 1-y_i\left(w^T x_i+b\right)\right)\right)\ +\ \frac{1}{2}\|\ \boldsymbol{w}\ \|_2^2$$

Hinge loss



Logistic Regression:

$$\underset{w,\ b}{Min}-\sum_{i=1}^{m}\ y_i\log\left(p_i\right)+\left(1-y_i\right)\log\left(1-p_i\right)\ +\ C\|\ w\ \|_2^2$$

Log loss

$$p(\ y=1|\ x)\ =\ \frac{1}{1+\ exp(\ -(\ w^T x\ +\ b)\ )}$$

Soft-Margin SVMs - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, LinearSVC(C=0.1, loss="hinge"))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

→ **primal**

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[0.97826087 0.97826087 0.97826087 0.95652174 0.97826087 1.
 0.97777778 0.97777778 0.95555556 1.        ]
0.9780676328502415
```

Soft-Margin SVMs - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, SVC(C=0.1, kernel='linear'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

→ **dual**

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[0.97826087 0.97826087 0.97826087 0.95652174 0.97826087 1.
 0.97777778 0.97777778 0.95555556 1.        ]
0.9780676328502415
```

Soft-Margin SVMs - hyperparameter tuning

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
pipe = make_pipeline(preprocess,
                     GridSearchCV(SVC(),
                                  param_grid = {"kernel":['linear'],
                                                "C":np.logspace(-3, 3, 20)},
                                  return_train_score=True))
pipe.fit(dev_X, dev_y)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
Best score: 0.9758241758241759
Best alpha: {'C': 0.0379269019073225, 'kernel': 'linear'}
Test score: 0.9824561403508771
```

Soft-Margin SVMs - prediction

Prediction: $$sign\left(\sum_i \alpha_i y_i \left(x \cdot x_i\right) + b\right)$$

```
grid_search_results.best_estimator_.dual_coef_
```

```
array([[-0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 ,
        -0.03177964, -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 ,
        -0.0379269 , -0.0379269 , -0.03527029, -0.0379269 , -0.0379269 ,
        -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 ,
        -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 ,
        -0.0379269 , -0.0379269 , -0.00737361, -0.0379269 , -0.0379269 ,
        -0.0379269 , -0.0379269 , -0.0379269 , -0.0379269 ,  0.01497148,
         0.00997104,  0.0379269 ,  0.0379269 ,  0.0379269 ,  0.02522742,
         0.0379269 ,  0.0379269 ,  0.0379269 ,  0.02246217,  0.0379269 ,
         0.0379269 ,  0.0379269 ,  0.0379269 ,  0.0379269 ,  0.0379269 ,
         0.0379269 ,  0.0379269 ,  0.00028438,  0.0379269 ,  0.0379269 ,
         0.0379269 ,  0.0379269 ,  0.0379269 ,  0.0379269 ,  0.0379269 ,
         0.0379269 ,  0.03464645,  0.0379269 ,  0.0379269 ,  0.0379269 ,
         0.0379269 ,  0.02857075,  0.0379269 ,  0.0379269 ,  0.01414365,
         0.0379269 ]])
```

```
grid_search_results.best_estimator_.support_vectors_
```

```
array([[ 0.32910211,  0.75680107,  0.28967589, ...,  0.12899967,
         0.42798088,  0.77697663],
       [ 0.27813844,  0.66528001,  0.22134988, ...,  0.59486965,
        -0.36464082, -0.28818525],
       [ 0.28663239,  2.49335462,  0.19871174, ..., -0.74220784,
         0.51992499, -1.24621394],
       ...,
       [ 0.24699397,  0.68640025,  0.23246315, ...,  0.25466197,
         0.40895795,  0.48474423],
       [ 1.05675009, -1.397464  ,  0.93506863, ..., -0.47157416,
        -1.77867793, -1.41112052],
       [-0.27679932,  0.36490318, -0.24293775, ...,  0.06003866,
        -0.54218808, -0.11935232]])
```

```
grid_search_results.best_estimator_.support_vectors_.shape
```

```
(71, 30)
```

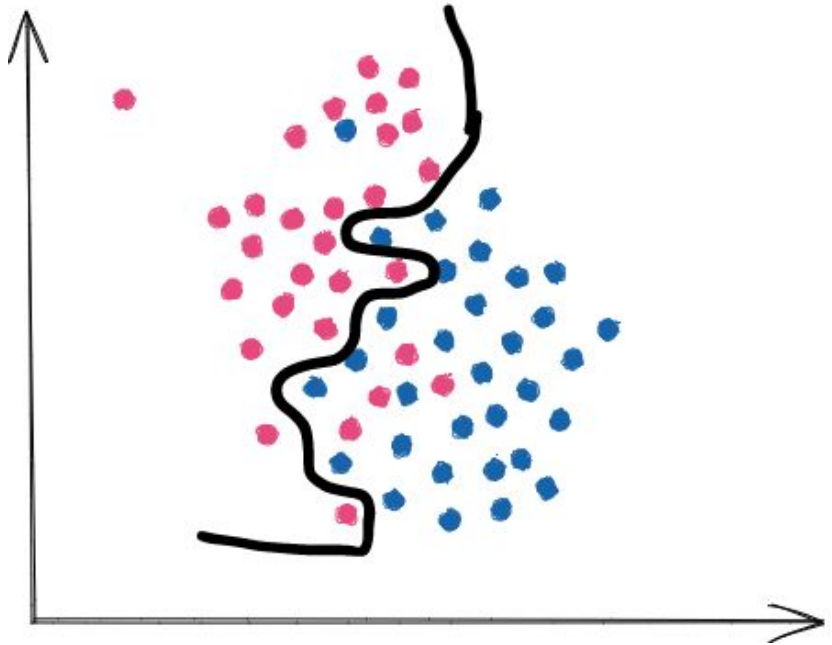
Hard-margin SVMs - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, SVC(C=1e8, kernel='linear'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[0.97826087 0.95652174 0.93478261 0.93478261 1.         1.
 0.95555556 0.93333333 0.91111111 0.95555556]
0.9559903381642514
```

Kernel SVMs

- Non-linear version of SVMs
- $\boldsymbol{\phi}(\boldsymbol{x})$ projects data to high-dimensional space
- Kernel function $\mathbb{K}(\boldsymbol{x};\boldsymbol{x})$ estimates inner product between two points in the projected space
- Kernel trick



Soft-Margin SVMs (dual)

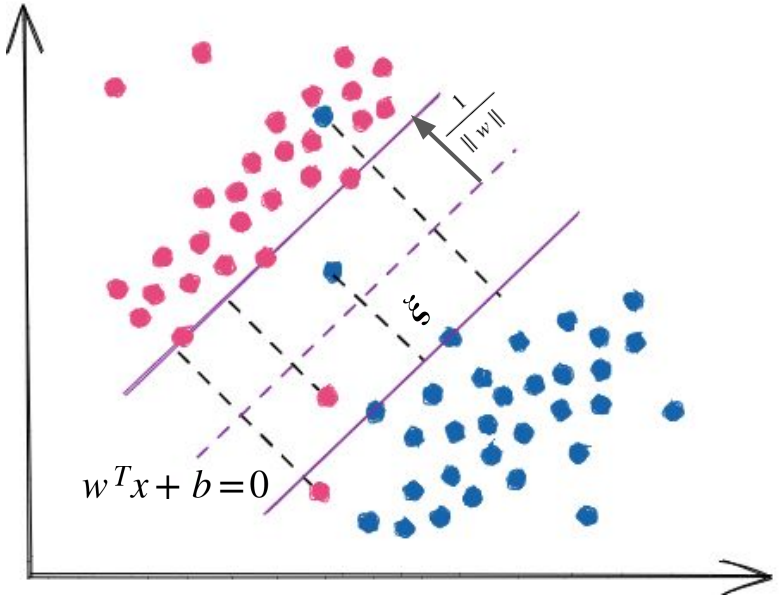
$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \right)$$

$$s.t. \sum_{i=1}^{m} \alpha_j y_j = 0$$

$$0 \leq \alpha_i \leq C \ \forall i$$

Prediction: $sign\left(\sum_i \alpha_i y_i \left(x \cdot x_i \right) + b \right)$

support vectors



Kernel SVMs

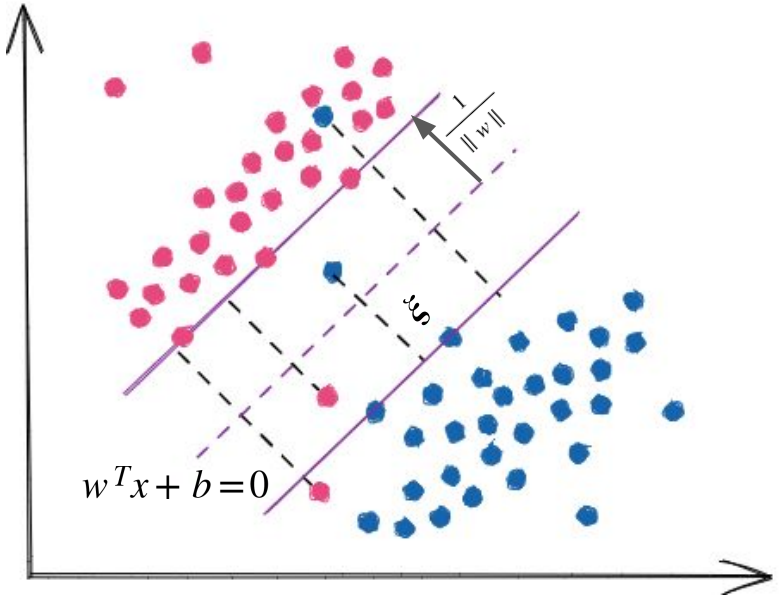
$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \right)$$

$$s.t. \sum_{i=1}^{m} \alpha_j y_j = 0$$

$$0 \leq \alpha_i \leq C \ \forall i$$

Prediction: $sign\left(\sum_i \alpha_i y_i \left(\phi(x) \cdot \phi(x_i) \right) + b \right)$

support vectors



Kernel SVMs

$$\underset{\alpha}{Max} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \mathbb{K}(x_i \cdot x_j)$$

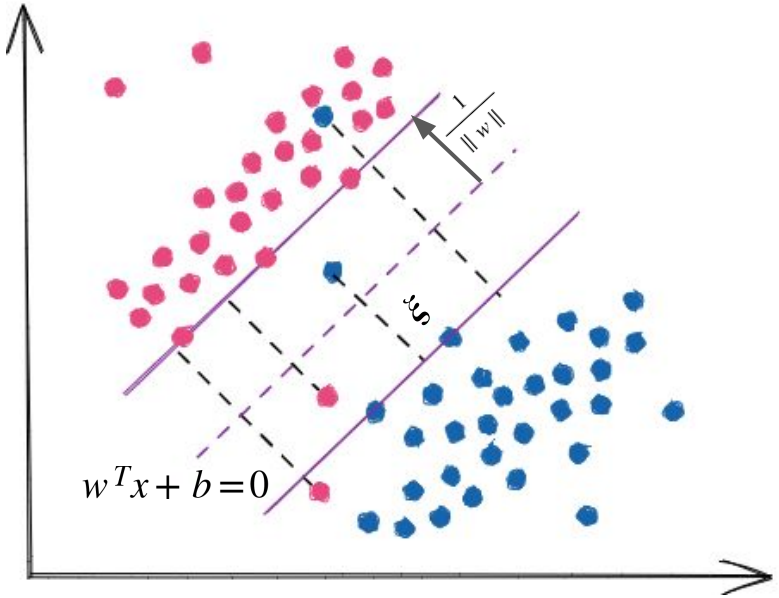
Kernel trick

$$s.t. \sum_{i=1}^{m} \alpha_j y_j = 0$$

$$0 \leq \alpha_i \leq C \ \forall i$$

Prediction: $sign\left(\sum_i \alpha_i y_i \mathbb{K}(x \cdot x_i) + b\right)$

support vectors



Kernel functions

$$\mathbb{K}\left(x_i, x_j \right) = \phi\left(x_i \right)^T \phi\left(x_j \right)$$

- If $\mathbb{K}(\boldsymbol{x}, \boldsymbol{x})$ is symmetric and positive definite, then it implies there exists a $\boldsymbol{\phi}(\boldsymbol{x})$
- Kernel function allows you to ***not*** define $\boldsymbol{\phi}(\boldsymbol{x})$ explicitly (**kernel trick**)

Kernel functions

- Kernels use either inner product or distances (in most cases)
- Feature scaling is important

Linear kernel

$$\mathbb{K}(x_i, x_j) = x_i^T x_j$$

Polynomial kernel

$$\mathbb{K}(x_i, x_j) = (x_i^T x_j + c)^p$$

RBF(Gaussian) kernel

$$\mathbb{K}(x_i, x_j) = exp(-\gamma \| x_i - x_j \|^2)$$

Kernel v.s. Explicit features

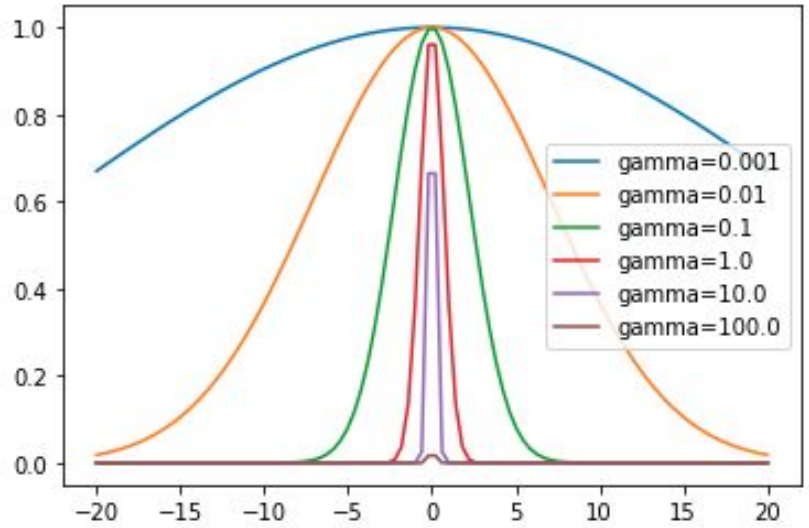
Assume:

N - # of samples, **c** - polynomial order, **d** - # of features

	Explicit features	**kernel**
Feature expansion	$O(Nd^c)$	N/A
Inner product	$O(d^c)$	$O(d*c)$
Example	$\phi(x_i) = \left(x_i, \sqrt{2}x_i, 1\right)$ $\phi(x_j) = \left(x_j, \sqrt{2}x_j, 1\right)$	$\mathbb{K}(x_i, x_j) = \left(x_i x_j + 1\right)^2$
	Polynomial features	**Polynomial kernel**

RBF Kernel

- The closer points in the projected space will have larger values
- Gamma controls how the function value decays with distance
- Considered a universal kernel since it projects to infinite dimensional space
- Can learn anything



Kernel SVMs - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, SVC(C=1.0, kernel='rbf', gamma=1.0))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[0.63043478 0.63043478 0.63043478 0.63043478 0.63043478 0.64444444
 0.62222222 0.62222222 0.62222222 0.62222222]
0.6285507246376811
```

Kernel SVMs - example

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, SVC(C=1.0, kernel='rbf', gamma=0.1))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

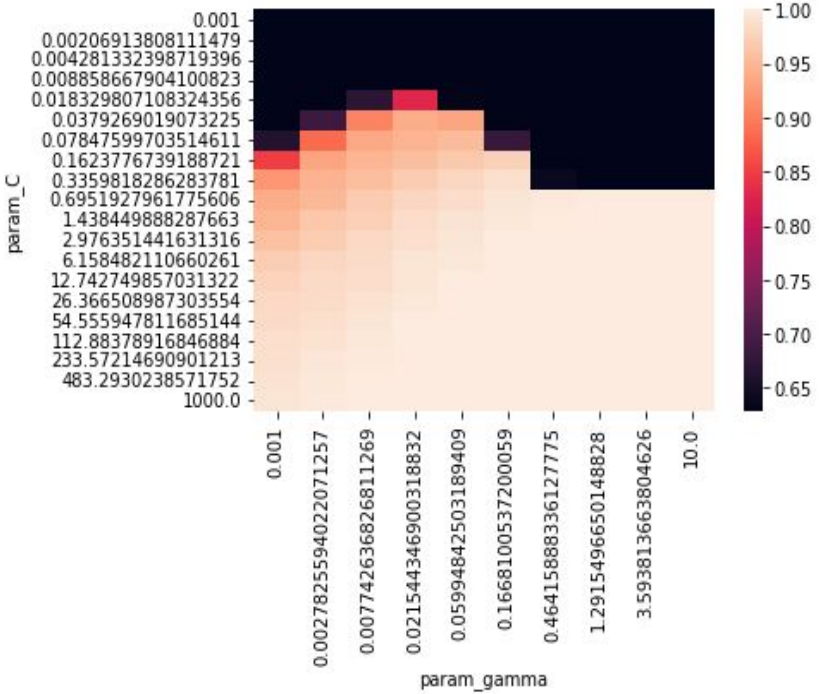
```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
[0.93478261 0.97826087 0.97826087 0.95652174 0.97826087 0.97777778
 0.97777778 0.95555556 0.97777778 0.86666667]
0.9581642512077295
```

Kernel SVMs - hyperparameter tuning

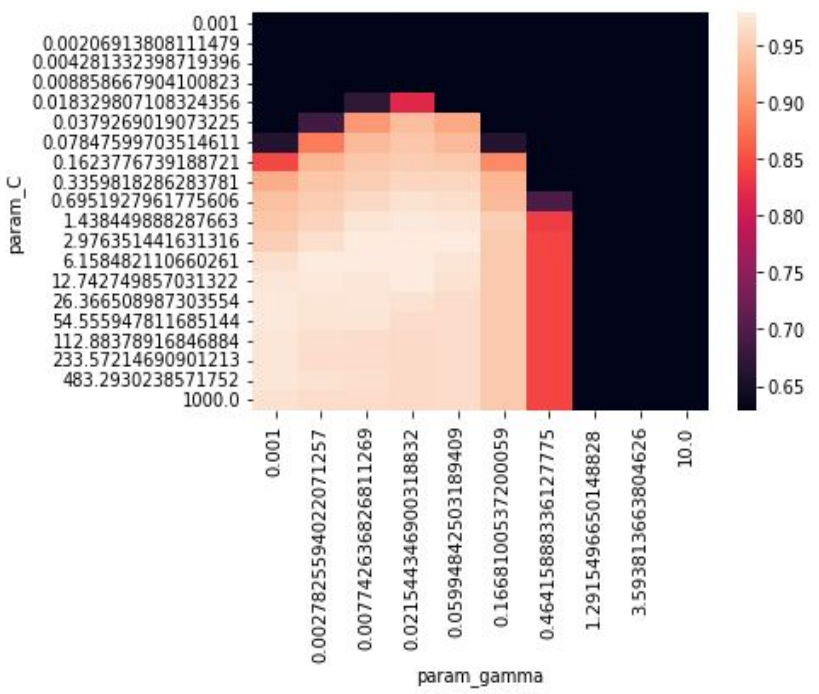
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
feature_names = data.feature_names
bc_df = pd.DataFrame(data.data, columns = feature_names)
target = pd.Series(data.target)
print("Class distribution:")
print(target.value_counts())
print("Dataset size:")
print(bc_df.shape)
dev_X, test_X, dev_y, test_y = train_test_split(bc_df, target,
                                                test_size=0.2, random_state=42)
pipe = make_pipeline(preprocess,
                     GridSearchCV(SVC(),
                                  param_grid = {"kernel":['rbf'],
                                                "C":np.logspace(-3, 3, 20),
                                                "gamma":np.logspace(-3, 1, 10)},
                                  return_train_score=True))
pipe.fit(dev_X, dev_y)
grid_search_results = pipe.named_steps["gridsearchcv"]
print(f"Best score:", grid_search_results.best_score_)
print(f"Best alpha:", grid_search_results.best_params_)
print(f"Test score:", pipe.score(test_X, test_y))
```

```
Class distribution:
1    357
0    212
dtype: int64
Dataset size:
(569, 30)
Best score: 0.9780219780219781
Best alpha: {'C': 2.976351441631316, 'gamma': 0.007742636826811269, 'kernel': 'rbf'}
Test score: 0.9824561403508771
```

Kernel SVMs - hyperparameter tuning



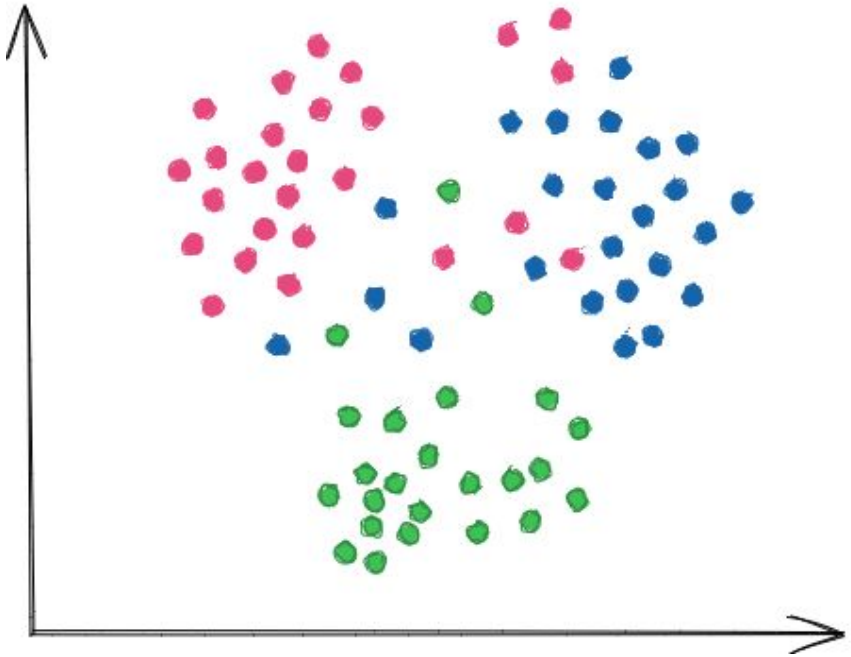
Train performance



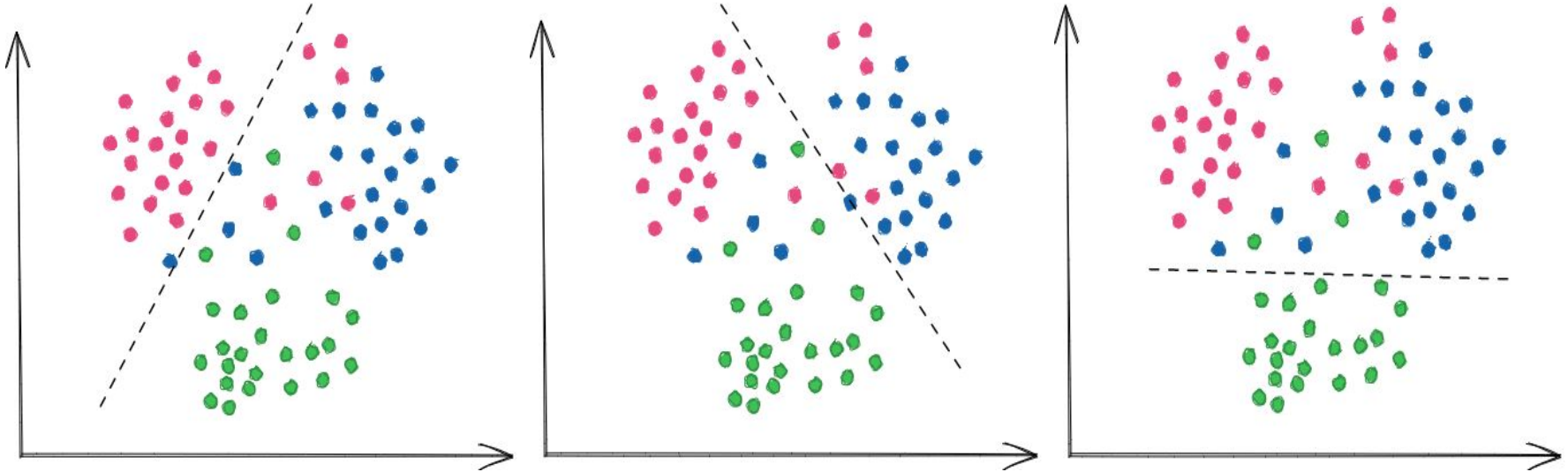
Validation performance

Multi-class classification

Multi-class classification



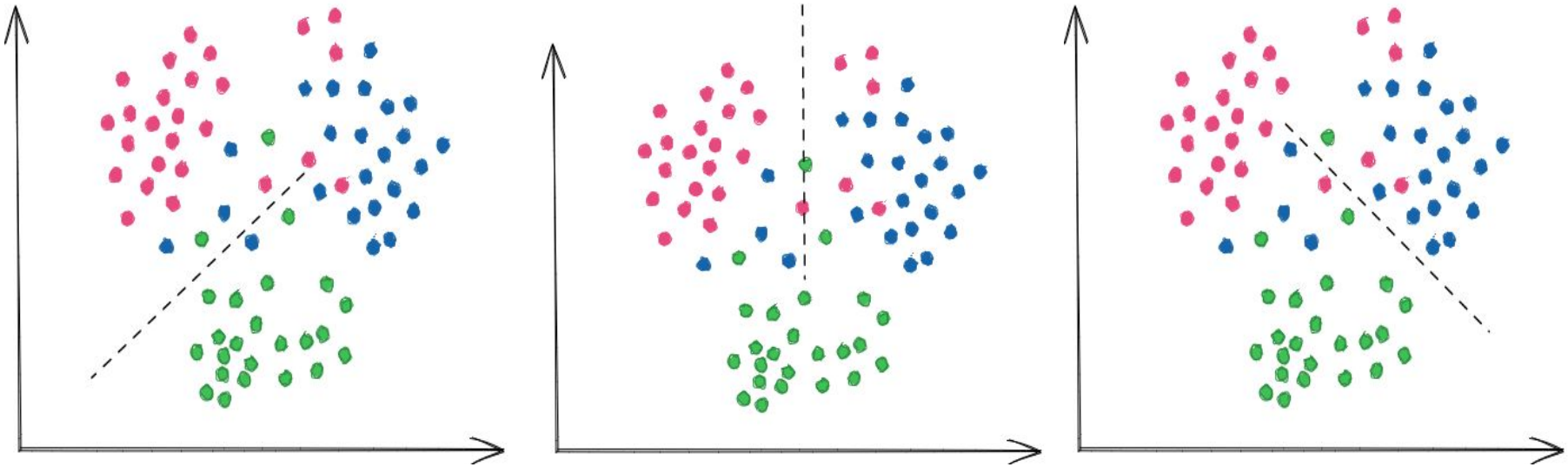
Leveraging binary classifiers



One v.s. Rest (OVR)

Prediction: $\hat{y} = \underset{i \in Y}{argmax}\left(\boldsymbol{w}_i{}^T \boldsymbol{x} + b_i \right)$ **(class with the highest score)**

Leveraging binary classifiers



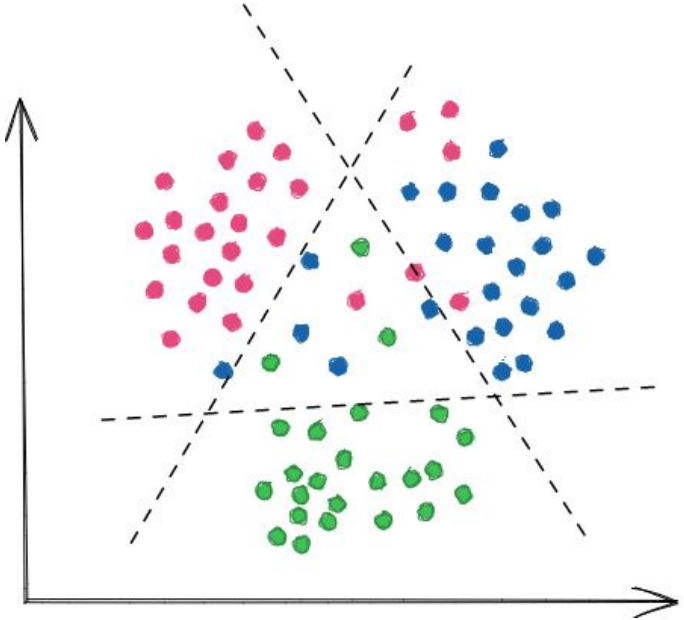
One v.s. One (OVO)

$$ClassPrediction(i, j, x) = \begin{cases} i \text{ if } \mathbf{w}_{ij}{}^{T}\boldsymbol{x} + b_{ij} \geq 0 \\ j \text{ otherwise} \end{cases}$$

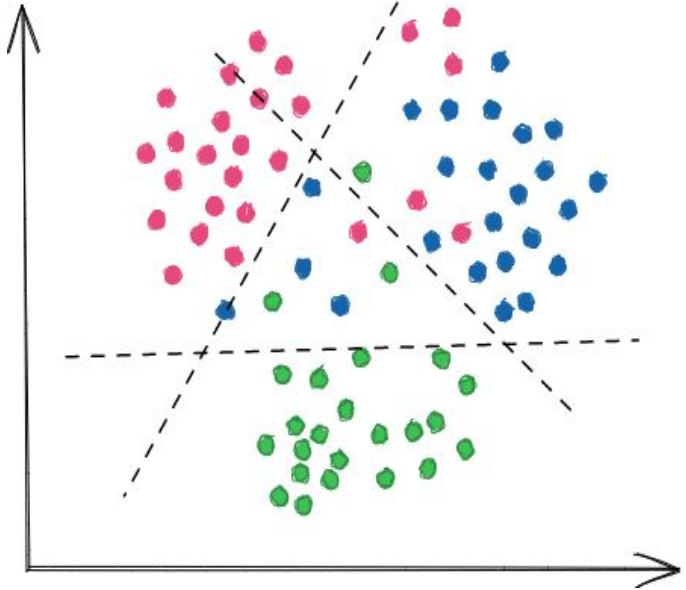
$$\hat{y} = mode\left(\underset{i,j \in Y}{ClassPrediction(x, i, j)}\right)$$

(most commonly predicted class)

OVR v.s. OVO



One v.s. Rest (OVR)



One v.s. One (OVO)

OVR v.s. OVO

One v.s. Rest (OVR)

- N binary classifiers
- Trains on imbalanced datasets
- Rarely a region of uncertainty
- Strategy applicable to most binary classifiers

One v.s. One (OVO)

- N(N-1)/2 binary classifiers
- Trains on balanced datasets (if original dataset is balanced w.r.t. all classes)
- Generally has a region of uncertainty
- Strategy applicable to most binary classifiers

Multinomial Logistic Regression

$$\ln(p(y=1|\boldsymbol{x})) = \left(\boldsymbol{w}_1{}^T\boldsymbol{x} + b_1\right) - \ln(z)$$

$$\ln(p(y=2|\boldsymbol{x})) = \left(\boldsymbol{w}_2{}^T\boldsymbol{x} + b_2\right) - \ln(z)$$

$$\ldots$$

$$\ln(p(y=K|\boldsymbol{x})) = \left(\boldsymbol{w}_K{}^T\boldsymbol{x} + b_K\right) - \ln(z)$$

$$z = \sum_{i=1}^{K} exp\left(\boldsymbol{w}_i{}^T\boldsymbol{x} + b_i\right)$$

Multinomial Logistic Regression

$$Pr(y=i|\boldsymbol{x}) = \frac{exp\left(\boldsymbol{w}_i{}^{\mathrm{T}}\boldsymbol{x}+b_i\right)}{\sum_{j=1}^{K} exp\left(\boldsymbol{w}_j{}^{T}\boldsymbol{x}+b_j\right)}$$

$$\underset{w,\ b}{Min} - \sum_{i=1}^{m} \log\left(p(y=y_i|\ \boldsymbol{x}_i, \boldsymbol{w}, b\right)$$

$$\hat{y} = \underset{i \in Y}{argmax}\left(\boldsymbol{w}_i{}^{T}\boldsymbol{x}+b_i\right)$$

Multiclass classification - example

```
def quality_to_label(x):
    if x <= 4:
        return 'poor'
    elif x > 4 and x <=6:
        return 'good'
    elif x > 6 and x <=8:
        return 'better'
    else:
        return 'best'
data_path = "data/winequality-red.csv"
wine_df = pd.read_csv(data_path)
wine_df["label"] = wine_df["quality"].apply(lambda x: quality_to_label(x))
print(wine_df.shape)
print(wine_df.columns)
wine_df.label.value_counts()
```

```
(1599, 13)
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'label'],
      dtype='object')
```

```
good      1319
better     217
poor        63
Name: label, dtype: int64
```

```
feature_names = [
    'fixed acidity',
    'volatile acidity',
    'citric acid',
    'residual sugar',
    'chlorides',
    'free sulfur dioxide',
    'total sulfur dioxide',
    'density',
    'pH',
    'sulphates',
    'alcohol',
]
wine_feature_df = wine_df[feature_names]
wine_target = wine_df["label"]
print("Dataset size:")
print(wine_feature_df.shape)
```

```
Dataset size:
(1599, 11)
```

https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009?select=winequality-red.csv

OVR v.s. OVO - Example

```
dev_X, test_X, dev_y, test_y = train_test_split(wine_feature_df, wine_target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, SVC(C=0.1, kernel='linear', decision_function_shape='ovr'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
[0.828125   0.828125   0.828125   0.828125   0.828125   0.828125
 0.828125   0.8203125  0.8203125  0.82677165]
0.8264271653543307
```

```
dev_X, test_X, dev_y, test_y = train_test_split(wine_feature_df, wine_target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, SVC(C=0.0001, kernel='linear', decision_function_shape='ovo'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
```

```
[0.828125   0.828125   0.828125   0.828125   0.828125   0.828125
 0.828125   0.8203125  0.8203125  0.82677165]
0.8264271653543307
```

Multinomial Logistic Regression - Example

```
dev_X, test_X, dev_y, test_y = train_test_split(wine_feature_df, wine_target,
                                                test_size=0.2, random_state=42)
preprocess = make_column_transformer((StandardScaler(), feature_names))
pipe = make_pipeline(preprocess, LogisticRegression('l2', C=1.0, multi_class='multinomial'))
scores = cross_val_score(pipe, dev_X, dev_y, cv=10, error_score="raise")
print(scores)
print(np.mean(scores))
pipe.named_steps["logisticregression"].fit(dev_X, dev_y).coef_.shape
```

```
[0.8515625  0.8359375  0.8515625  0.875      0.8515625  0.8046875
 0.8515625  0.8359375  0.84375    0.85826772]
0.8459830216535433
(3, 11)
```

Questions?